

Frontiers
in
Artificial
Intelligence
and
Applications

SOFTWARE ENGINEERING: EVOLUTION AND EMERGING TECHNOLOGIES

Edited by
Krzysztof Zieliński
Tomasz Szmuc

IOS
Press

VISIT...

LANZAROTE
Caliente.COM

SOFTWARE ENGINEERING: EVOLUTION AND EMERGING TECHNOLOGIES

Frontiers in Artificial Intelligence and Applications

FAIA covers all aspects of theoretical and applied artificial intelligence research in the form of monographs, doctoral dissertations, textbooks, handbooks and proceedings volumes. The FAIA series contains several sub-series, including “Information Modelling and Knowledge Bases” and “Knowledge-Based Intelligent Engineering Systems”. It also includes the biannual ECAI, the European Conference on Artificial Intelligence, proceedings volumes, and other ECCAI – the European Coordinating Committee on Artificial Intelligence – sponsored publications. An editorial panel of internationally well-known scholars is appointed to provide a high quality selection.

Series Editors:

J. Breuker, R. Dieng, N. Guarino, J.N. Kok, J. Liu, R. López de Mántaras,
R. Mizoguchi, M. Musen and N. Zhong

Volume 130

Recently published in this series

- Vol. 129. H. Fujita and M. Mejri (Eds.), New Trends in Software Methodologies, Tools and Techniques
- Vol. 128. J. Zhou et al. (Eds.), Applied Public Key Infrastructure
- Vol. 127. P. Ritrovato et al. (Eds.), Towards the Learning Grid
- Vol. 126. J. Cruz, Constraint Reasoning for Differential Models
- Vol. 125. C.-K. Looi et al. (Eds.), Artificial Intelligence in Education
- Vol. 124. T. Washio et al. (Eds.), Advances in Mining Graphs, Trees and Sequences
- Vol. 123. P. Buitelaar et al. (Eds.), Ontology Learning from Text: Methods, Evaluation and Applications
- Vol. 122. C. Mancini, Cinematic Hypertext –Investigating a New Paradigm
- Vol. 121. Y. Kiyoki et al. (Eds.), Information Modelling and Knowledge Bases XVI
- Vol. 120. T.F. Gordon (Ed.), Legal Knowledge and Information Systems – JURIX 2004: The Seventeenth Annual Conference
- Vol. 119. S. Nascimento, Fuzzy Clustering via Proportional Membership Model
- Vol. 118. J. Barzdins and A. Caplinskas (Eds.), Databases and Information Systems – Selected Papers from the Sixth International Baltic Conference DB&IS’2004
- Vol. 117. L. Castillo et al. (Eds.), Planning, Scheduling and Constraint Satisfaction: From Theory to Practice
- Vol. 116. O. Corcho, A Layered Declarative Approach to Ontology Translation with Knowledge Preservation
- Vol. 115. G.E. Phillips-Wren and L.C. Jain (Eds.), Intelligent Decision Support Systems in Agent-Mediated Environments

ISSN 0922-6389

Software Engineering: Evolution and Emerging Technologies

Edited by

Krzysztof Zieliński

AGH University of Science and Technology, Kraków, Poland

and

Tomasz Szmuc

AGH University of Science and Technology, Kraków, Poland

IOS
Press

Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

© 2005 The authors.

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without prior written permission from the publisher.

ISBN 1-58603-559-2

Library of Congress Control Number: 2005932064

Publisher

IOS Press

Nieuwe Hemweg 6B

1013 BG Amsterdam

Netherlands

fax: +31 20 687 0019

e-mail: order@iospress.nl

Distributor in the UK and Ireland

IOS Press/Lavis Marketing

73 Lime Walk

Headington

Oxford OX3 7AD

England

fax: +44 1865 750079

Distributor in the USA and Canada

IOS Press, Inc.

4502 Rachael Manor Drive

Fairfax, VA 22032

USA

fax: +1 703 323 3668

e-mail: iosbooks@iospress.com

LEGAL NOTICE

The publisher is not responsible for the use which might be made of the following information.

PRINTED IN THE NETHERLANDS

Preface

The capability to design quality software and implement modern information systems is at the core of economic growth in the 21st century. Nevertheless, exploiting this potential is only possible when adequate human resources are available and when modern software engineering methods and tools are used.

The recent years have witnessed rapid evolution of software engineering methodologies, including the creation of new platforms and tools which aim to shorten the software design process, raise its quality and cut down its costs. This evolution is made possible through ever-increasing knowledge of software design strategies as well as through improvements in system design and code testing procedures. At the same time, the need for broad access to high-performance and high-throughput computing resources necessitates the creation of large-scale, interactive information systems, capable of processing millions of transactions per seconds. These systems, in turn, call for new, innovative distributed software design and implementation technologies.

The purpose of this book is to review and analyze emerging software engineering technologies, focusing on the evolution of design and implementation platforms as well as on novel computer systems related to the development of modern information services. The eight chapters address the following topics covering a wide spectrum of contemporary software engineering:

1. **Software Engineering Processes** – software process maturity, process measurement and evaluation, agile software development, workflow management in software production,
2. **UML-based Software Modeling** – UML 2.0 features, usability of UML modeling, exception modeling, business environment elaboration with UML,
3. **Software Process Methodologies** – extreme programming, test-driven development, increasing source code quality, software complexity analysis,
4. **Technologies for SOA** – Grid systems and services, distributed component platforms, configuration management, system and application monitoring,
5. **Requirements Engineering** – gathering, analyzing and modeling requirements, analyzing and modeling business processes, requirements management,
6. **Knowledge Base System and Prototyping** – knowledge base system engineering, integrating ontologies, modular rule-based systems,
7. **Software Modeling and Verification** – modeling of rule-based systems, modeling and verification of reactive systems,
8. **Selected Topics in Software Engineering** – this part covers 8 selected topics related to various aspects of software engineering.

We believe that the presented topics are interesting for software engineers, project managers and computer scientists involved in the computer software development process. We would like to express our thanks to all authors, colleagues, and reviewers who have supported our efforts to prepare this book.

Krzysztof Zieliński
Tomasz Szmuc

Reviewers

Marian Bubak	<i>AGH University of Science and Technology</i>
Zbigniew Czech	<i>Silesian University of Technology</i>
Janusz Gorski	<i>Gdansk University of Technology</i>
Zbigniew Huzar	<i>Wroclaw University of Technology</i>
Andrzej Jaszkiwicz	<i>Poznan University of Technology</i>
Jacek Kitowski	<i>AGH University of Science and Technology</i>
Henryk Krawczyk	<i>Gdansk University of Technology</i>
Ludwik Kuźniarz	<i>School of Engineering, Ronneby, Sweden</i>
Antoni Ligeza	<i>AGH University of Science and Technology</i>
Jan Madey	<i>Warsaw University</i>
Lech Madeyski	<i>Wroclaw University of Technology</i>
Jan Magott	<i>Wroclaw University of Technology</i>
Zygmunt Mazur	<i>Wroclaw University of Technology</i>
Marek Milosz	<i>Lublin University of Technology</i>
Edward Nawarecki	<i>AGH University of Science and Technology</i>
Jerzy Nawrocki	<i>Poznan University of Technology</i>
Krzysztof Sacha	<i>Warsaw University of Technology</i>
Andrzej Stasiak	<i>Military University of Technology</i>
Stanisław Szejko	<i>Gdansk University of Technology</i>
Zdzisław Szyjewski	<i>University of Szczecin</i>
Marek Valenta	<i>AGH University of Science and Technology</i>
Bartosz Walter	<i>Poznan University of Technology</i>
Jan Werewka	<i>AGH University of Science and Technology</i>
Kazimierz Wiatr	<i>AGH University of Science and Technology</i>
Bogdan Wiszniewski	<i>Gdansk University of Technology</i>
Robert Chwastek	<i>ComArch S.A.</i>
Jarosław Deminet	<i>Computerland S.A.</i>
Jacek Drabik	<i>Motorola</i>
Janusz Filipiak	<i>ComArch S.A.</i>
Piotr Fuglewicz	<i>TiP Sp. z o.o.</i>
Bartosz Nowicki	<i>Rodan Systems S.A.</i>
Marek Rydzy	<i>Motorola</i>
Andrzej Wardzinski	<i>PROKOM Software S.A.</i>
Lilianna Wierchoń	<i>Computerland S.A.</i>

Contents

Preface	v
<i>Krzysztof Zieliński and Tomasz Szmuc</i>	
Reviewers	vi
1. Software Engineering Processes	
Software Process Maturity and the Success of Free Software Projects	3
<i>Martin Michlmayr</i>	
The UID Approach – the Balance Between Hard and Soft Methodologies	15
<i>Barbara Begier</i>	
Agile Software Development at Sabre Holdings	27
<i>Marek Bukowy, Larry Wilder, Susan Finch and David Nunn</i>	
Workflow Management System in Software Production & Maintenance	39
<i>Paweł Markowski</i>	
Architecture of Parallel Spatial Data Warehouse: Balancing Algorithm and Resumption of Data Extraction	49
<i>Marcin Gorawski</i>	
2. UML-Based Software Modeling	
Data Modeling with UML 2.0	63
<i>Bogumiła Hnatkowska, Zbigniew Huzar and Lech Tuzinkiewicz</i>	
Usability of UML Modeling Tools	75
<i>Anna Bobkowska and Krzysztof Reszke</i>	
On Some Problems with Modelling of Exceptions in UML	87
<i>Radosław Klimek, Paweł Skrzyński and Michał Turek</i>	
The ISMS Business Environment Elaboration Using a UML Approach	99
<i>Andrzej Białas</i>	
3. Software Process Methodologies	
Preliminary Analysis of the Effects of Pair Programming and Test-Driven Development on the External Code Quality	113
<i>Lech Madeyski</i>	
Codespector – a Tool for Increasing Source Code Quality	124
<i>Mariusz Jadach and Bogumiła Hnatkowska</i>	

Software Complexity Analysis for Multitasking Systems Implemented in the Programming Language C <i>Krzysztof Trocki and Mariusz Czyz</i>	135
--	-----

4. Technologies for SOA

Grid Enabled Virtual Storage System Using Service Oriented Architecture <i>Lukasz Skitał, Renata Słota, Darin Nikolov and Jacek Kitowski</i>	149
Internet Laboratory Instructions for Advanced Software Engineering Course <i>Ilona Bluemke and Anna Derezińska</i>	160
Configuration Management of Massively Scalable Systems <i>Marcin Jarzab, Jacek Kosinski and Krzysztof Zielinski</i>	172
Interoperability of Monitoring Tools with JINEXT <i>Włodzimierz Funika and Arkadiusz Janik</i>	184
TCPN-Based Tool for Timing Constraints Modelling and Validation <i>Sławomir Samolej and Tomasz Szmuc</i>	194

5. Requirements Engineering

Requirement Management in Practice <i>Michał Godowski and Dariusz Czyrnek</i>	209
Implementation of the Requirements Driven Quality Control Method in a Small IT Company <i>Stanisław Szejko, Maciej Brochocki, Hubert Lyskawa and Wojciech E. Kozłowski</i>	221
System Definition and Implementation Plan Driven by Non-Linear Quality Function Deployment Model <i>Tomasz Nowak and Mirosław Głowacki</i>	233
Change Management with Dynamic Object Roles and Overloading Views <i>Radosław Adamus, Edgar Głowacki, Tomasz Serafiński and Kazimierz Subieta</i>	245

6. Knowledge Base System and Prototyping

The Process of Integrating Ontologies for Knowledge Base Systems <i>Jarosław Koźlak, Anna Zygmunt, Adam Łuszpaj and Kamil Szymański</i>	259
Designing World Closures for Knowledge-Based System Engineering <i>Krzysztof Goczyła, Teresa Grabowska, Wojciech Waloszek and Michał Zawadzki</i>	271
The Specifics of Dedicated Data Warehouse Solutions <i>Anna Zygmunt, Marek A. Valenta and Tomasz Kmiecik</i>	283

Formal Approach to Prototyping and Analysis of Modular Rule-Based Systems	294
<i>Marcin Szpyrka and Grzegorz J. Nalepa</i>	
Selection and Testing of Reporting Tools for an Internet Cadastre Information System	305
<i>Dariusz Król, Małgorzata Podyma and Bogdan Trawiński</i>	
7. Software Modeling and Verification	
Use-Cases Engineering with UC Workbench	319
<i>Jerzy Nawrocki and Łukasz Olek</i>	
Conceptual Modelling and Automated Implementation of Rule-Based Systems	330
<i>Grzegorz J. Nalepa and Antoni Ligeza</i>	
Model Management Based on a Visual Transformation Language	341
<i>Michał Śmialek and Andrzej Kardas</i>	
Exploring Bad Code Smells Dependencies	353
<i>Błażej Pietrzak and Bartosz Walter</i>	
Modeling and Verification of Reactive Software Using LOTOS	365
<i>Grzegorz Rogus and Tomasz Szmuc</i>	
8. Selected Topics in Software Engineering	
Evaluation of Software Quality	381
<i>Krzysztof Sacha</i>	
CMMI Process Improvement Project in ComputerLand	389
<i>Lilianna Wierzchón</i>	
Development of Safety-Critical Systems with RTCP-Nets Support	394
<i>Marcin Szpyrka</i>	
ATG 2.0: The Platform for Automatic Generation of Training Simulations	400
<i>Maciej Dorsz, Jerzy Nawrocki and Anna Demuth</i>	
RAD Tool for Object Code Generation: A Case Study	406
<i>Marcin Graboń, Jarosław Małek, Marcin Surkont, Paweł Woroszczuk, Rafał Fitrzyk, Andrzej Huzar and Andrzej Kaliś</i>	
KOTEK: Clustering of the Enterprise Code	412
<i>Andrzej Gąsienica-Samek, Tomasz Stachowicz, Jacek Chrzęszcz and Aleksy Schubert</i>	
Inference Mechanisms for Knowledge Management System in E-Health Environment	418
<i>Krzysztof Goczyła, Teresa Grabowska, Wojciech Waloszek and Michał Zawadzki</i>	

Open Source – Ideology or Methodology? <i>Krzysztof Dorosz and Sebastian Gurgul</i>	424
Author Index	431

1. Software Engineering Processes

This page intentionally left blank

Software Process Maturity and the Success of Free Software Projects

Martin MICHLMAYR

Department of Computer Science and Software Engineering
University of Melbourne
Victoria, 3010, Australia
*e-mail: martin@michlmayr.org*¹

Abstract. The success of free software and open source projects has increased interest in utilizing the open source model for mature software development. However, the ad hoc nature of open source development may result in poor quality software or failures for a number of volunteer projects. In this paper, projects from SourceForge are assessed to test the hypothesis that there is a relationship between process maturity and the success of free software and open source projects. This study addresses the question of whether the maturity of particular software processes differs in successful and unsuccessful projects. Processes are identified that are key factors in successful free software projects. The insights gained from this study can be applied as to improve the software process used by free software projects.

Keywords. Process maturity, quality improvement, free software, open source

Introduction

The development model employed in free software and open source projects is unique in many respects. Free software and open source projects are traditionally characterized by two factors that have an impact on quality and project management [8], [7]. The first factor is that they are performed by volunteers, the second that the participants are globally distributed. In addition to these two important characteristics, the development model is also crucially determined by the nature of free software and open source itself; all source code is available and shared with others. Raymond suggests that this open exchange leads to a high level of peer review and user involvement [11], [13]. It has been shown in software engineering research that peer review contributes to the quality of software [1]. This may offer an explanation for the high levels of quality found in some successful free software and open source projects, such as Linux and Apache.

While there has been an increase in research carried out on free software and open source, only a small amount of attention has been given to investigating which traditional software engineering insights can be mapped to the open source development model [6]. Since volunteers working on free software and open source are often driven by different motivations to those of software companies developing proprietary software [3], it is not clear whether all strategies postulated in software

¹ Present address: Centre for Technology Management, University of Cambridge, UK

engineering can be applied to open source projects. However, this issue is of high relevance if techniques are to be found that further improve the success of the open source development model.

This paper investigates whether the maturity of the processes employed by distributed, volunteer projects is connected to their success. While most software engineering research on open source has concentrated on successful projects (for example Apache [9], Debian [2], [14], GNOME [5] and Linux [15]), the observation can be made that there is a large number of unsuccessful open source projects that have not been studied in detail to date [4]. Our hypothesis is that there is a link between process maturity and the success of a project, especially where processes related to coordination and communication are concerned. During the investigation of this hypothesis, key areas of the software process linked to the success of free software and open source projects are identified. These factors give an indication of the importance of various processes and will contribute to the creation of more successful free software and open source projects.

1. Methodology

This paper is based on an empirical case study involving 80 projects hosted on SourceForge. This is currently the largest hosting site for free software and open source projects with over 95,000 projects and more than one million registered developers. The maturity of these randomly chosen projects were evaluated using a simple assessment designed for this purpose. One central goal during the development of this assessment was to create a general mechanism for evaluating process maturity rather than one specific to the present research question. The assessment form can therefore be used by other researchers and practitioners to judge the maturity of any open source project; it is included in Appendix A. Due to the generalizable nature of the assessment it is described before the actual case study.

1.1. Project Assessment

An assessment has been developed for this study that can be used to evaluate some crucial aspects of the maturity of free software and open source projects. In particular, the assessment focuses on the maturity of processes related to coordination and communication since these are important components in distributed projects [7]. The assessment makes use of empirical data typically available in open source projects. The validity of the assessment is assumed to be high since most questions can be answered objectively and only some require personal judgement. While neither the inter-rater nor the re-test reliability have been statistically verified, both showed reliable results in this case study.

The assessment incorporates insights from both software engineering and open source. The assessment is based on quality models which postulate the importance of high quality and disciplined processes. In particular, emphasis was put on processes which might be important in open source projects. Specifically, the distributed and volunteer nature of free software and open source projects was taken into account. Since free software and open source benefit from a strong community surrounding a project [11], [16], processes to attract volunteers and integrate them into the project are vital.

The assessment is grouped into five categories taking different aspects of free software projects into account. The majority of questions can be addressed with a simple “yes” or “no” answer since they concern the presence or availability of a process or piece of infrastructure. For some questions, a finer grained measure is beneficial and zero, one or two points are given for these questions (more points indicating higher maturity). The whole assessment with a description of the questions is available in Appendix A

1.1.1. Version Control

The first question relates to version control and configuration management in the project. The use of version control tools, such as CVS [18], allows multiple volunteers to work on the same code with little coordination. Furthermore, the wide availability of the code repository encourages volunteers to review code, remove defects or add new functionality. Therefore, this question distinguishes whether a version control system is used at all, as well as whether it is publicly available or not.

1.1.2. Mailing Lists

Mailing lists are also important in free software projects because they create an effective and fast medium for communication and coordination. Furthermore, mailing lists are often used to answer questions posed by users and developers and hence encourage more people to get involved and contribute to the project. This question differentiates between the presence of no mailing list at all, one mailing list, or multiple specialized lists. Dedicated mailing lists for different topics help to keep the focus. There can be lists for users, developers and announcements. There is also a question about the availability of mailing list archives. Mailing list archives give users and developers the opportunity to follow previous discussions and to find answers which have previously been given. This reduces the time spent by developers answering questions that have already been addressed, and it lowers the barrier of entry for prospective developers.

1.1.3. Documentation

The third category is about documentation, both for users and developers. There might be an important link between user documentation and the success of a project since it is possible that users will not be able to use the software if it is not well documented. Dedicated developer documentation might encourage new volunteers to contribute by reducing the learning curve associated with getting involved in a project. It could also increase the maintainability and quality of the source code if everyone adheres to the same coding standards outlined in the developer documentation.

1.1.4. Systematic Testing

The next three questions relate to systematic testing and the availability of important infrastructure and processes. The first question asks whether the project publishes release candidates before issuing stable releases. Release candidates are an indication that a project has a clear release plan. The next question concerns the availability of an automatic test suite comprising regression or unit tests, and the third one checks the presence of an defect or issue tracking system. Such a system might encourage users to report defects and hence enables developers to remove them and improve the quality of

the software. It may also be a good mechanism to capture feedback and ensure that valuable input is not lost.

1.1.5. Portability

Finally, the last question is about the portability of the software. Software that can be ported to a wide variety of platforms attracts more users who might contribute to the project. Additionally, porting software to a new platform often highlights defects which only occur under special circumstances. This question differentiates between three levels of portability. A piece of software might be aimed specifically at one platform, can support a set of hard-coded platforms, or supports multiple platforms through a system which automatically identifies the characteristics of a system.

1.2. Case Study

The previously described assessment was employed to evaluate the maturity of 80 projects from SourceForge. SourceForge was chosen because it is a very popular hosting site for free software and open source projects. At the time of writing, SourceForge hosts more than 95,000 projects and has over one million registered developers. It provides various types of infrastructure for projects, such as CVS, mailing lists and issue tracking systems. While this hosting site was chosen for the data collection, the findings of this research are intended to also apply to open source projects not hosted on SourceForge. Since the assessment form used for the case study are described in detail, the findings here can be replicated by other researchers.

Since it is the aim of this study to investigate whether the maturity of the employed processes is linked to the success of a project, 40 successful and 40 unsuccessful projects were randomly selected for this investigation. SourceForge provides elaborate statistics about projects and their download data was taken to define success. While downloads do not equate or necessarily imply quality or even success [4], it is a fairly good measure of fitness for purpose because users of open source software must actively download this software rather than use pre-installed applications on their computer systems. Additionally, the advantage of using downloads as a measure is that it is objective and independent – the users determine this measure rather than the researcher.

As a first step, 40 successful projects were selected from one of SourceForge's statistics page which lists the 100 top downloads. Since comparability is a vital criterion in this case study, projects were excluded which do not create a piece of software but rather assemble a large collection of software created at other places. Additionally, projects for which no source code was available were not taken into account since this would not allow a complete assessment. Following these criteria, 33 projects out of the top 100 were excluded. From those remaining, 40 projects were chosen randomly.

In order to select unsuccessful projects, the statistics on downloads was employed. Specifically, the percentile value for a given project from September 2003 was taken because this indicates how a project is placed in relation to others. Our randomly selected successful projects had a mean percentile of 97.66% ($\sigma = 2.78$). Taking this figure into account, an upper limit of 75% was arbitrarily chosen as the criterion of an unsuccessful project as this is considerable less than the average placement of successful projects. During the selection of unsuccessful projects it became evident that

they were on average much smaller in size than the successful ones. While this observation is interesting and should be pursued in greater detail in the future, it is problematic for this case study since the two groups had to be of similar characteristics and *only* differ in their success. If the groups differed in other ways it could not be clearly argued that the findings of this study are related only to project success. Hence, unsuccessful projects were selected randomly until the group of unsuccessful projects had similar characteristics as the one consisting of successful projects. In particular, the factors of age, lines of code (LOC) and development status were taken into account (see Table 1). Development status is based on SourceForge's categorization of the project; lines of code of a project was determined with the popular program SLOCCount by David A. Wheeler [20], [2]. At the end, the groups were adequately comparable, with neither age ($t(78) = 1.466$; $p = 0.147$), lines of code ($t(72) = 1.47$; $p = 0.146$) or development status ($W = 921$; $p = 0.089$) showing a significant difference. However, the groups highly differed in their success ($t(40) = 16.04$; $p < 0.001$).

Table 1. Means and standard deviation of successful and unsuccessful projects

		Age	LOC	Status	Ranking
Mean	successful	1163	126500	4.43	97.66%
	unsuccessful	1066	78140	4.11	41.18%
Standard deviation	successful	298	167995	1.06	2.78%
	unsuccessful	291	123109	0.98	22.09%

All 80 projects were assessed by two graduate students, each focusing on 40 random projects. The assessment was performed on a blind basis, i.e. the students were not aware whether a project was deemed successful or unsuccessful. While they were graduate students of computer science with a solid background in the general field, they had little experience with free software and open source projects, so reducing the likelihood that their assessment would be biased by the recognition of successful and popular open source projects. Since the assessment has been designed in a way to ensure high validity even if little familiarity with open source is present, the selection of these assessors strengthens the methodological foundations of this study.

The assessment was carried out in the first week of October 2003. After the data gathering and assessment was completed and verified, various statistical tests were performed. The chi-square test was used for nominal data while the Wilcoxon test was employed for ordinal data. All tests were performed with the statistics package GNU R, version 1.8.0, and were carried out with $\alpha = 0.05$ (the probability of rejecting the statistical hypothesis tested when, in fact, that hypothesis is true). The null hypothesis for all tests was that there is no difference between successful and unsuccessful projects, and all tests were two-sided.

2. Results and Findings

The data for each question of the assessment form was analyzed, and the results will be presented individually in the following section, followed by a discussion and a prospective of further research.

2.1. Results

In total, nine statistical tests were carried out, corresponding to the questions which comprise the assessment.² As expected, given the number of tests, some tests revealed significant differences in the two groups of successful and unsuccessful projects while others did not. Version control is one aspect where the groups differ significantly ($W = 1046.5$, $p = 0.004$). While the successful group shows 1.75 points on average ($\sigma = 0.59$) out of 2, the unsuccessful projects only score 1.275 points ($\sigma = 0.85$). Similarly, the availability of mailing lists differ between the two groups ($W = 1113.5$, $p < 0.001$). Not only do successful projects make better use of mailing lists, gaining 1.55 points on average ($\sigma = 0.78$) out of 2 as compared to 0.925 points ($\sigma = 0.86$) of unsuccessful projects, they also provide archives of their discussions in more cases. A total of 80% of successful projects offer mailing archives while only half of the unsuccessful projects do so.

Table 2. Availability of documentation

		User	Developer
Successful	available	23 (57.5%)	10 (25.0%)
	not available	17 (42.5%)	30 (75.0%)
Unsuccessful	available	24 (60.0%)	5 (12.5%)
	not available	16 (40.0%)	35 (87.5%)

The third category of the assessment is related to documentation in the project (Table 2). The groups do not differ significantly either in the availability of user documentation ($\chi^2 = 0$; $p = 1$) or of documentation aimed at developers ($\chi^2 = 1.31$; $p = 0.25$). While about 60% of projects of projects offer user documentation, only 25% of successful and 12.5% of unsuccessful projects offer developer documentation. The fourth category, testing, reveals mixed results. Significantly more successful projects offer release candidates ($\chi^2 = 5.16$; $p = 0.023$), and make use of a defect tracking system ($\chi^2 = 24.96$; $p < 0.001$). However, the groups do not differ significantly in the use of regression or unit test suites ($\chi^2 = 0.37$; $p = 0.54$). Table 3 shows exactly how much the different facilities are employed by the two groups. In summary, most successful projects use Source-Forge's defect tracking system, more than half of the successful projects provide release candidates but few projects use test suites. On the hand, only about 30% of unsuccessful projects use the defect tracking system and make release candidates available.

Table 3. Presence of testing facilities

		Release Candidates	Automatic Test Suite	Defect Tracking
Successful	used	22 (55.0%)	5 (12.5%)	35 (87.5%)
	not used	18 (45.0%)	35 (87.5%)	5 (12.5%)
Unsuccessful	used	11 (27.5%)	8 (20.0%)	12 (30.0%)
	not used	29 (72.5%)	32 (80.0%)	28 (70.0%)

² The results are given in standard statistical notations. W refers to the Wilcoxon test used for ordinal data whereas χ^2 denotes the chi-square test used for nominal data. p is the probability; if it is lower than α , the null hypothesis is rejected.

Finally, there is no major difference between the groups with regards to portability ($W = 793$, $p = 0.95$). The two groups score almost equally, with successful projects gaining an average of 1.325 points ($\sigma = 0.76$) and unsuccessful projects scoring 1.375 points ($\sigma = 0.62$). These results will be discussed in more detail in the following section.

2.2. Discussion

A statistical analysis of the data gained through the evaluation of 40 successful and 40 unsuccessful projects shows that 5 out of 9 tests show significant differences between the two groups. In those cases where major differences can be found, the successful projects have been found to employ a more mature process than unsuccessful project. Due to the high number of tests which show differences between the groups, it can be concluded that the maturity of the processes employed by a project is linked to its success.

This paper has therefore shown the existence of a relationship between the processes used by a project and its success. However, the present study did not investigate the nature of this relationship. In particular, no conclusions on a possible causality between process maturity and success can be drawn from the present study. Further in-depth work on the qualitative aspects of the relationship between process maturity and success have to be carried out.

Finally, although this paper clearly shows that the maturity of the processes play an important role, there may be other factors which have not been considered in the present study.

2.2.1. Version Control

Version control tools, such as CVS in the case of projects hosted on SourceForge, are used more widely in successful than in unsuccessful projects. Furthermore, a higher percentage of version control repositories are available publicly. In free software projects, CVS and similar tools play important roles related to coordination. Having a private version control repository limits the access of prospective contributors. On the other hand, a publicly available CVS repository allows contributors to monitor exactly what other people are working on. This allows multiple people to work together efficiently in a distributed way. In a similar way to defect tracking systems, public version control systems may attract more volunteers since users see what needs to be done and how they can get involved.

2.2.2. Mailing Lists

Similarly to version control tools, mailing lists have an important function for coordination in a project. Archived mailing lists also replace documentation to some degree since users can find answers to questions which have been asked before. Prospective volunteers can follow previous discussions about the development and can so easily learn how to take part in the project. In both, version control tools and mailing lists, it is not clear from the present findings whether a successful project requires these types of infrastructure to flourish, or whether the implementation of the infrastructure has attracted more volunteers and so led to more success. Our assumption is that there is no clear causality and that both affect each other. However, it is clear that mailing lists and version control tools are key factors in successful open source projects.

2.2.3. Documentation

The presence of user and development documentation is not a good indication for the success of a project. While more than half of the projects in this case study offer documentation for users, a much smaller number provides developer documentation. This difference suggests that the findings must be interpreted differently for user and developer documentation. Since a larger number of projects provide documentation that do not, it can be argued that user documentation is an important factor in free software projects. However, this factor alone does not determine success. Developer documentation, on the other hand, is not very common in free software projects. A reason for this might be that prospective developers find most answers to their questions from other sources, such as mailing list archives or the source code. They can derive the coding style used in the project directly from the source code if there is no guide about the coding style, and can follow the development process on the mailing lists. They can effectively learn the rules by observing them in action [19]. This suggests that written documentation is not necessary because it is available in indirect form already. Competent developers will be able to find all information from other sources and by directly observing the process. While this explanation seems very plausible, there is one remaining question. It would be interesting to study the code quality of different free software and open source projects to investigate whether the availability of documentation for developers leads to higher code quality. It is also possible that the availability of good developer documentation will attract more potential developers who will contribute to a project.

2.2.4. Systematic Testing

The analysis of processes involving testing shows that successful projects make more use of release candidates and defect tracking systems than unsuccessful projects. There is no difference in the presence of automated test suites. The availability of release candidates has been taken as an indication of a well defined release plan. While it would be revealing to qualitatively study and compare the release strategies of different projects, the present findings suggest that a clear release strategy coupled with testing contributes to the success of a project. Furthermore, defect tracking systems play a crucial role. While the successful and unsuccessful groups in this case study do not differ significantly in their size, successful projects are likely to receive more feedback from users. This feedback has to be captured properly so it can later be prioritized and analyzed. Neither successful nor unsuccessful projects make much use of automated test suites. One reason for this may be the nature of open source itself. Raymond argues that open source projects benefit from a greater community surrounding them that provides defect reports [11]. Once a regression occurs it is very likely that one of the users will notice it quickly and report it. This might be a reason why little time is spent on implementing automated tests. However, it remains to be seen whether the use of test suites could lead to even higher quality. The use of automated tests would allow volunteer to spend less time on tracking down regressions and they could devote their time to the removal of other defects or the implementation of new features.

2.2.5. Portability

This last assessed factor is very high in both groups. A likely reason for this is that the Unix philosophy has always promoted portability [12]. Connected to this is the

availability of software, such as autoconf [17], [10], which makes it relatively easy for a piece of software to automatically determine the characteristics of a platform and to adapt to it. Another factor which could contribute to portability is the diverse nature of volunteers participating in free software projects.

2.2.6. Summary

Several factors have been identified which are linked to the success of a project. Projects which have incorporated version control, effective communication mechanism such as mailing lists and various testing strategies into their software development process are more successful than other projects. These findings clearly support the hypothesis that free software and open source projects profit from the use of mature processes. While this study has focused on processes connected to coordination and communication, two factors that are crucial in distributed projects, the open source process might also benefit from other processes used in traditional software engineering projects. This finding is in contrast to the suggestion of some open source developers that their development model is unique and will not benefit from process maturity or traditional insights [6].

2.3. Further Work

The present paper clearly shows that free software and open source projects benefit from a highly mature process. This study supports the notion that some software engineering insights can be applied to the open source development model. The results of this paper are of substantial value and demonstrate the importance of further research in this area to address some of the questions raised in this paper. While this study showed that developer documentation does not contribute to the success of a project, this paper brings forward the suggestion that solid developer documentation can lead to higher quality source code. A study which compares the code quality of projects with and without good developer documentation can shed more light on this question. Furthermore, while the use of automated test suites in free software projects is not currently common, they may offer certain benefits to the participants of a volunteer project. Another question related to testing is whether projects using defect tracking systems have a higher defect removal rate than projects using less reliable techniques to track defects. Finally, a qualitative approach comparing different release strategies could lead to specific guidelines being developed that promote successful strategies for delivering tested and stable open source software for end-users.

3. Conclusions

This paper started with the observation that free software and open source projects are different to traditional software engineering projects in many respects. This is mainly because free software and open source projects are typically performed by volunteers who are distributed all over the world. This important difference to other software engineering projects, such as those deployed by companies, raises the question whether traditional insights can be applied to open source projects. It has been proposed that some fundamental insights from software quality models, such as the demand for mature processes, can be applied to open source projects. Some processes praised by

software engineering, such as the requirement for requirements specifications and other formal guides, could be incompatible with the motivation of volunteer participants. However, others can be unified with the open source development model and contribute to success and quality. In particular, processes connected to coordination and communication appear to be vital for free software projects.

In order to investigate this hypothesis, a case study has been conducted involving randomly chosen projects from SourceForge. In the case study, 40 successful and 40 unsuccessful projects were assessed regarding their maturity in several aspects using an assessment developed in this paper. The results of various statistical analyses clearly showed that the maturity of some processes are linked to the success of a project. This study identified the importance of the use of version control tools, effective communication through the deployment of mailing lists, and found several effective strategies related to testing. The identification of processes employed by successful free software projects is of substantial value to practitioners since they give an indication of which areas deserve attention.

The findings of this paper emphasize the importance of applying software engineering insights to the open source development model. Further research has been suggested to explore some of the results of this paper in more detail, such as a thorough comparison of code quality and a qualitative evaluation of release strategies. The identification of important processes, together with more research in this area, will contribute to higher quality and more successful free software and open source projects.

Acknowledgements

Thanks are given to Deepika Trehan and S Deepak Viswanathan, two graduate students at the University of Melbourne, for assessing the maturity of the 80 projects in this case study. Thanks are also extended to Mike Ciavarella and Paul Gruba for their continuous support throughout this case study and to the anonymous reviewers for their extensive comments. This work has been funded in part by Fotango.

A. Assessment Questions

A.1. Version Control

1. Does the project use version control?
 - No, the project does not use any version control (0 points)
 - Yes, it is used but the repository is private (1 point)
 - Yes, it is used and the repository available to the public (2 points)

A.2. Mailing Lists

1. Are mailing lists available?
 - No, the project has no mailing list at all (0 points)
 - Yes, there is one mailing list (1 point)
 - Yes, there are multiple, specialized mailing lists (2 points)
2. Are mailing list archives available?

A.3. Documentation

1. Does the project have documentation for users?
2. Does the project have documentation for developers?

A.4. Systematic Testing

1. Are release candidates available?
2. Does the source code contain an automatic suite?
3. Does the project use a defect tracking system?

A.5. Portability

1. Is the software portable?
 - No, it has been written specifically for a single platform (0 points)
 - Yes, it supports a limited number of platforms (1 point)
 - Yes, it automatically determines the properties of a platform (2 points)

References

- [1] Michael E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976.
- [2] Jesús M. González-Barahona, Miguel A. Ortuño Pérez, Pedro de las Heras Quirós, José Centeno González, and Vicente Matellán Olivera. Counting Potatoes: The Size of Debian 2.2. *Upgrade*, II(6):60–66, December 2001.
- [3] Alexander Hars and Shaosong Ou. Why is open source software viable? A study of intrinsic motivation, personal needs, and future returns. In *Proceedings of the Sixth Americas Conference on Information Systems (AMCIS 2000)*, pages 486–490, Long Beach, California, 2000.
- [4] James Howison and Kevin Crowston. The perils and pitfalls of mining Source-Forge. In *Proceedings of the International Workshop on Mining Software Repositories (MSR 2004)*, pages 7–11, Edinburgh, UK, 2004.
- [5] Stefan Koch and Georg Schneider. Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal*, 12(1):27–42, 2002.
- [6] Bart Massey. Why OSS folks think SE folks are clue-impaired. In *3rd Workshop on Open Source Software Engineering*, pages 91–97. ICSE, 2003.
- [7] Martin Michlmayr. Managing volunteer activity in free software projects. In *Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track*, pages 93–102, Boston, USA, 2004.
- [8] Martin Michlmayr and Benjamin Mako Hill. Quality and the reliance on individuals in free software projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pages 105–109, Portland, OR, USA, 2003. ICSE.
- [9] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.
- [10] Andy Oram and Mike Loukides. *Programming With GNU Software*. O’Reilly & Associates, 1996.
- [11] Eric S. Raymond. *The Cathedral and the Bazaar*. O’Reilly & Associates, Sebastopol, CA, 1999.
- [12] Eric S. Raymond. *The Art Of Unix Programming*. Addison-Wesley, 2003.
- [13] Eric S. Raymond and W. Craig Trader. Linux and open-source success. *IEEE Software*, 16(1):85–89, 1999.
- [14] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Martin Michlmayr. Evolution of volunteer participation in libre software projects: Evidence from Debian. In Marco Scotto and Giancarlo Succi, editors, *Proceedings of the First International Conference on Open Source Systems*, pages 100–107, Genova, Italy, 2005.
- [15] Stephen R. Schach, Bo Jin, David R. Wright, Gillian Z. Heller, and A. Jefferson Offutt. Maintainability of the Linux kernel. *IEE Proceedings – Software*, 149(1):18–23, 2002.

- [16] Anthony Senyard and Martin Michlmayr. How to have a successful free software project. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pages 84–91, Busan, Korea, 2004. IEEE Computer Society.
- [17] Gary V. Vaughan, Ben Elliston, Tom Tromey, and Ian Lance Taylor. *GNU Autoconf, Automake, and Libtool*. SAMS, 2000.
- [18] Jennifer Vesperman. *Essential CVS*. O'Reilly & Associates, 2003.
- [19] Georg von Krogh, Sebastian Spaeth, and Karim R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, 2003.
- [20] David A. Wheeler. More than a gigabuck: Estimating GNU/Linux's size, 2001.
<http://www.dwheeler.com/sloc>

The UID Approach – the Balance between Hard and Soft Methodologies

Barbara BEGIER

*Institute for Control and Information Engineering,
Poznan University of Technology,
Pl. M. Skłodowskiej-Curie 5, 60-965 Poznan, Poland
e-mail: Barbara.Begier@put.poznan.pl*

Abstract. The UID approach (*User Involved Development*) is suggested to improve software development process. Its aim is to admit users as its subject. The UID approach shows an attempt to combine features of hard and soft methodologies. The user focus leads to users' involvement in the process. An initial questionnaire survey of all potential users is recommended. There are suggested specified forms of developers-users cooperation to provide a continual feedback between both sides. Business process modeling, software product assessment by all its users, competency-based selection to play various roles, construction of questionnaires – are examples of methods in use.

Introduction

The shockingly low percentage of software products which satisfy their users [9] is a motive for looking for new models of software production. Software developers seem to be unable to provide high quality software supporting public institutions and organizations. The universal approach to produce software for military applications, control devices, mobile communication, and public service, doesn't seem to be right.

There are observed two opposing trends in software engineering. The dominating one is represented by *hard methodologies*, focused on making software production fast and repeatable. The main criterion of success is a high productivity of developers. Users are told that all detected failures and arriving problems are caused by a growing complexity of a system and changes in requirements.

On the opposite side there are *soft methodologies* (including XP [2], Participatory Design [24] and other agile methods), which tend, first of all, to satisfy users – the quality has been the declared value for 30 years. The notion of a *user* is not well specified, so it is hard to say whose satisfaction is intended to guarantee.

A growing demand for a software process, which provides high quality products, cannot ignore economic constraints. There is a need to find a middle ground to balance between hard and soft methodologies, which arose from different aims and values. The UID approach (*User Involved Development*) is introduced to combine opposing characteristics of hard and soft methodologies. In the described proposal the quality is related to the level of various users' satisfaction from a product.

The described approach has been derived from the author's research on assessing the level of user's satisfaction from the software product. Research conclusions and author's personal observations show that direct users have not been asked for their

suggestions on the information system under development. Developers do not value those opinions. Views of indirect users are not taken into consideration at all.

It's been assumed that the quality of software tools has a positive or negative effect on the quality of users' work and her/his life – quality of a workplace is a part of the quality of life. The aim of the UID approach is to change the philosophy underlying the software process – its subject are users who are no more supposed just to adapt themselves to the delivered system. Specified factors and metrics play auxiliary roles. In the UID approach users are involved in the software process from its very beginning till withdrawing a product from use. It corresponds to human aspects in software process [1]. In the described approach the quality of indirect users' life is at least as much important as the satisfaction of direct users. Solutions already applied in business area [21] are adapted to improve management in the software process: management of human resources at producer's and purchaser's sides, techniques applied in market research, a continual process improvement [17] as recommended in the ISO 9000:2000, business process modelling, competency management.

Users' involvement and their assessment of a product improves a process, and, in consequence, a product. The UID approach is mainly intended for the software systems built by making use of public funds. Examples are like: public administration, national insurances, medical care, judiciary, etc. All social and ethical aspects of the undertaken project should be analyzed [3] at the very beginning.

1. Main Assumptions Underlying Hard and Soft Methodologies

Processes established in an organization are results of respected values and specified goals, as shown in the Figure 1, although employees are often not conscious of them. Implemented and enacted processes may support human values and a quality of life (quality of work is its essential part) or just help to complete a project. Processes usually follow defined aims of a software company, mostly financial only. So the absence of users in the software development process is due to the lack of respect for users' work, competencies, and expectations. Indeed, there are a lot of prejudices between developers and users. The former ones have an unjustified feeling of superiority over users and the latter sneer at alleged professionalism, which manifests rather in incomprehensible language than in high quality of products.

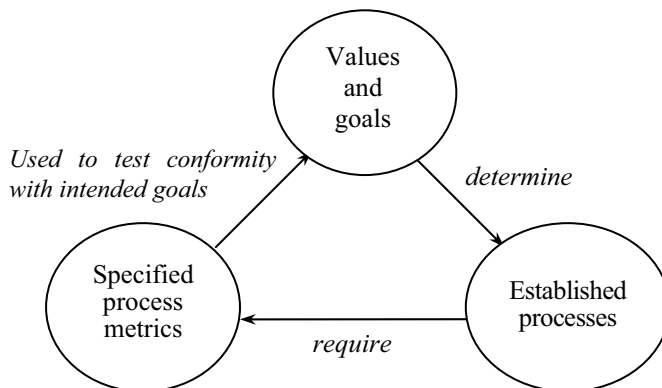


Figure 1. Context to establish and assess processes

In hard methodologies, the plan-driven software process is well structured, defined, organized, and controlled – plans are feasible, techniques learnt, requirements identified, practices accepted. Technical reviews and inspections are sufficient tools to make sure that a client will validate and accept the product. Technical factors and defined procedures are supposed to be the crucial elements in a whole process. Competencies of developers are reduced to technical skills and work in teams. Quality standards are only tools to manipulate clients [23].

Engineers try to specify and automate the software process as much as possible. Thus *design patterns* of data and procedures have appeared, which mean progress in routine applications, including banking or telecommunications. *Hard methodologies* [18] are based on some general assumptions, which may be expressed as following:

- Previous experiences are sufficient to build next products correctly.
- Technical standards and available tools provide enough mechanisms to fulfill any given requirements.
- The software process is well structured and it makes all required control procedures available and practically sufficient to provide quality.
- It is possible to apply the same process to any software product.
- There are always available experts to organize the software process properly.
- Required interpersonal communication relies heavily on documented know-ledge. Products descriptions and progress reports are precise enough to develop and modify software. People rotations off teams do not generate problems.
- All direct and indirect users are a monolithic group – the homogeneous whole.
- The direct communication between developers and a wide spectrum of users is not necessary and even impossible.
- Software corporations have enough economic power and forceful persuasion to continue their current policy of production.

Hard methodologies are suitable for the producer's market and this is the case. They are the right approach in military applications and also in routinely built systems. It's the well-known fact that the main stream of research and development in computer science and software engineering is driven by military orders. But it is discussible that quality management and process maturity assessment should be also based on the military commission (all work on the CMM model has been completely sponsored by the U.S. Department of Defense). The points of view of indirect users are not taken into consideration at all. The lack of users-developers cooperation results in transferring the yesterday's solutions into a new system. The real life example, related to the office maintaining land records, is shown in the Figure 2.

Unlike hard methodologies, the so called *soft* (also: *agile*, *ecological*) *methodologies* [2], [11], [14], [24] are characterized by the following fundamental assumptions:

- Each new kind of applications means an experimental work.
- Purchasers are able to build software by themselves or they are ready to delegate their representatives to cooperate with producers in an every day mode.
- Person-to-person communication is always available and all personnel shows the willingness to share their professional knowledge.

- No software methodologies are too difficult to be learnt and mastered by representatives of the purchaser's.
- If resources are exhausted then the process may be discontinued at any moment and a satisfying product is delivered to users.

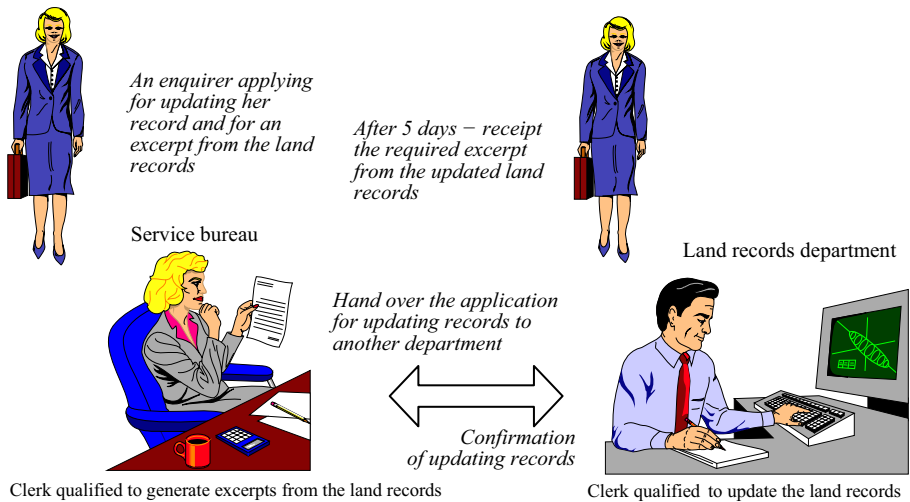


Figure 2. An example illustrating an effect of the lack of users' participation in the software development process

In practice, principles of these methodologies are applicable in teams less than 10 people mostly because of communication problems. Not all, developed in early phases, models are later really implemented, but some reflections on discussed solutions stay valuable in the future. The notion of *system borders* has been incorporated into the SSADM (*Structured Software Analysis and Design Methodology*). No complete specification is needed, if it is not a priori decided which parts of a system are definitely realized. Even semiformal methods are abandoned for any informal diagrams (*rich pictures*). All participants of a process try to work out definitions (*root definitions*) and other settings. The CATWOE (*Clients, Actors, Transformation process, Weltanschauung, Owners & Environment*) analysis is recommended – there are analysed requirements of all *stakeholders*, in other words all *interested parties* as used in the ISO 9000:2000, involved in the process. To sum up, results of the development process are unspecified in advance and undetermined.

2. The Guidelines of the UID Approach

Some terms, used in this section, need to be explained. The *customer* or a *purchaser* is an organization, which orders a software product, for example: a hospital, the Municipal Council, the tax office. The term *client* refers to the organization of the purchaser and all its personnel. The words *personnel* and *employees* are synonyms. The *direct user* means a clerk including a person holding a managerial position who

operates and uses the considered system at work. The *indirect user* depends indirectly on the system – he/she is a patient or a supplicant of the given organization, like a hospital or a register office. The term of *users' representatives* and *representative users* mean the same in this paper: a set of people who represent the broad spectrum of direct and indirect users – average people, as they really are.

The guidelines listed below are related to a software process, established to develop a new and untypical, so far, information system. The UID (*User Involved Development*) approach is described by the following statements:

- The *customer focus* covers requirements of *direct* and *indirect* users.
- Software product is a result of a process, which is a composition of established processes. The underlying values are agreed on and known for both sides.
- All required processes are identified, defined, established, and managed.
- An evolutionary life cycle model is followed.
- The software development cycle starts from the initial questionnaire survey to get learn all potential users, and their competencies.
- Some elements of agile methodologies and other user-oriented methods are incorporated into a standardized process. Numerous representative users are actively involved in the entire software development process. Forms of users' involvement are specified, agreed on, and planned in advance. There is a *continual feedback* between developers and users – it is referred to any development phase and to each particular form of users' involvement.
- Characteristics of users and developers include *personality types* and *competencies* to assign proper people to play roles in teams. Competency management including the competency-based recruitment and selection is provided, too.
- The plans and measures play an important role, concerning also various forms of users' involvement in the process.
- User-oriented quality standards are really enforced in software production.
- Business process modeling becomes an obligatory phase of each cycle.
- The QFD (*Quality Function Deployment*) technique is followed.
- The process of product assessment by a wide spectrum of users is established and cyclically repeated.
- The CRM (*Customer Relationship Management*) system is established to maintain proper relationships with client's representatives [6].
- All employees at the producer's side are involved in a continual improvement of processes – a system to collect proposals of improvements is established.
- The provided document management system is also a mean of improvement of internal and external communication.
- The technical and organizational infrastructure supports the suggested ideas.
- Some activities are automated by involving software agents.

The close cooperation between developers and users is the core of the UID approach. The earlier research [4], [5] has provided the author with users' opinions concerning software product and, indirectly, the process. Users are experts in their domain but so far they do not know all possibilities of modern information techniques, even those seemingly well known, like exchange of data and documents between systems, or mechanisms to speed up activities to enter data. The research revealed that

users had no opportunity to exchange their views on information system with developers. They even had an unpleasant feeling of being ignored and underestimated in their capabilities to influence the particular software process.

Users do not represent nowadays the same level of education and professional skills as they did ten years ago. They've become experienced in information techniques and mature enough to become partners in the software life cycle. On the base of agile methods [1] and of own research there are recommended various forms of cooperation with users as shown in the Figure 3. The presented set is not exhausted.

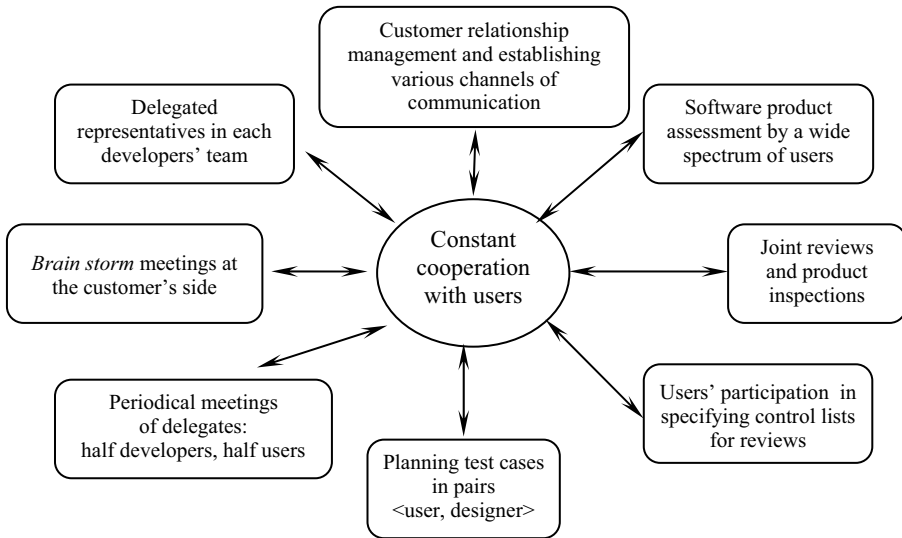


Figure 3. The recommended forms of cooperation with users

Some activities are supposed to be automated by involving software agents. Besides main tasks in software process there are a lot of minor activities, which are not planned in advance, for example: exchanging messages, organizing meetings, planning routine small tasks, collecting documents, management of a change, etc. Those small tasks are assigned to agents' activity.

3. A Software Development Cycle in the UID Approach

The UID approach may be combined with any evolutionary life cycle model, including the software staged life cycle model [19] or the spiral WinWin model [8]. The last one and the UID approach have in common: identifying stakeholders in each spiral cycle, learning win conditions (to be accepted) and negotiations. But users are not supposed to participate in later phases in the WinWin model.

The *initial questionnaire survey* is recommended to find out users' categories and competencies. Otherwise it would be hard to say who is really representative for the users' community. Suggested questions concern: age bracket, sex, skills in computing, personality type, personal preferences, and competencies.

An analysis of business processes and a discussion on their new models are an essential phase of each software development cycle as shown in the Figure 4. Users' representatives take part in all phases. They are actively present at any work concerning software under development. The cost of users' participation seems to be much less than the cost of failures and required software reconstructions.

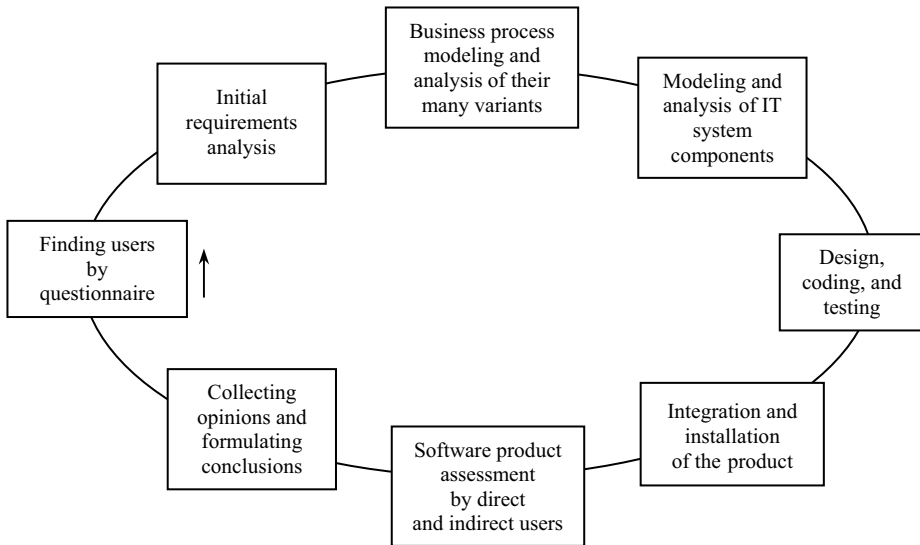


Figure 4. One iteration of software development cycle in the UID approach

There is a common practice to incorporate previous solutions into a new information system. For example, it takes three months to get from the centralized register office in France the required certificate of being able to marriage. An enquirer only applies on-line immediately. Developers have ignored real-life problems of indirect users.

To avoid wrong solutions, **business process modelling** is recommended as an obligatory phase in the software development cycle. Tools, applied at first to define and improve business processes according to the recommendations of the ISO 9000, are suggested to model processes, which can be supported by an information system. Representative users play the most important role to describe, model, discuss, and choose proper solutions. The suggested model for the case illustrated earlier in the Figure 2 is illustrated in the Figure 5 – an enquirer receives the required document at the same visit in the office. That model of the process of service was developed using the ARIS methodology [20] and Toolset.

The **software product assessment** by possibly wide spectrum of direct and indirect users helps to formulate requirements for the next increment. The process to assess a level of user's satisfaction from a software product is based on questionnaire method [4], [5] adapted from the market research.

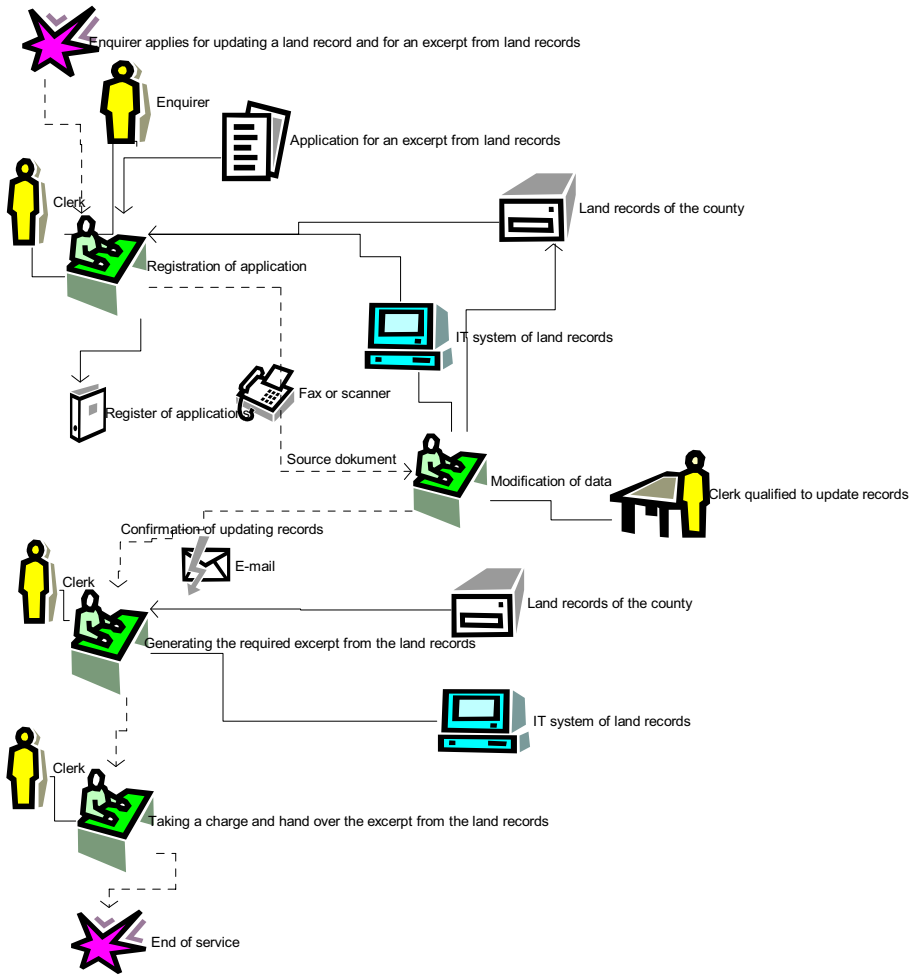


Figure 5. Scheme of one business process in the land records department

4. Permanent Feedback between Developers and Users

Forms of cooperation during one development phase provide four loops of feedback as shown in the Figure 6. Representative users are delegated to all teams established in software development process. To remove barriers between developers and users, the proper selection of representatives at the producer's side should take place. Reported figures show that capabilities of software developers are below average in the world's scale [22] and that's why agile methods are not applicable on the large scale. The rhythm of cooperation activities becomes regular and acceptable.

The most internal loop is related to the presence of users' delegates in developers' teams. The second one bears of meetings arranged in the half-half proportion of users and developers. The source of the next one is the brain storm meeting at the purchaser's side. Users' representatives take part in joint reviews and inspections, at least as co-authors of control lists. In each loop the corrective actions take place according to the

users' opinions and suggestions. Brain storm meeting as well as technical reviews are planned in advance.

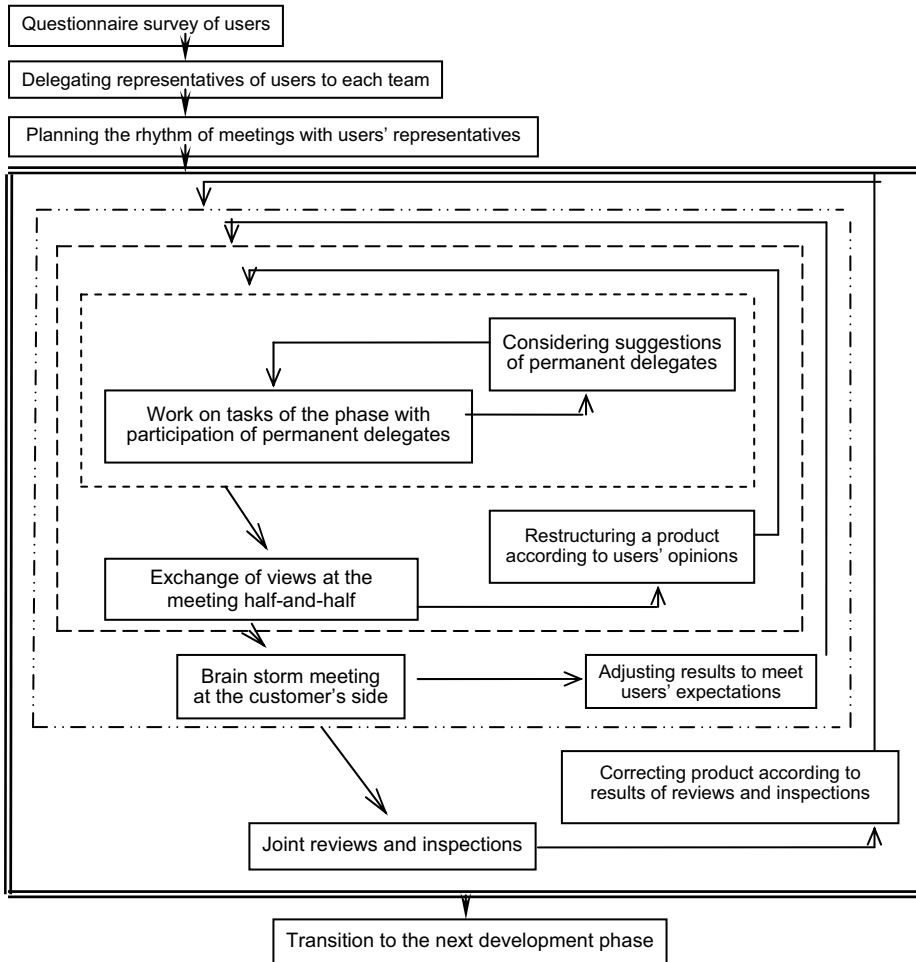


Figure 6. Iterations and permanent feedback from users in the development phase

5. Competency-Based Selection of Personnel

The software process is too complex to continue an impersonal view of software development. People factors have started to play a significant role in software management [12], [13]. These factors have been so far related mainly to the productivity of teams. There are at least five areas where employees demonstrate a substantial diversity: *staffing*, *culture*, *values*, *communications*, and *expectations* [22]. None of them is an isolated island. Only *staffing* refers directly to capabilities, skills and the number of developers. The meaning of the listed areas is still narrow – for example, there are distinguished just two organisational cultures: agile or plan-driven culture.

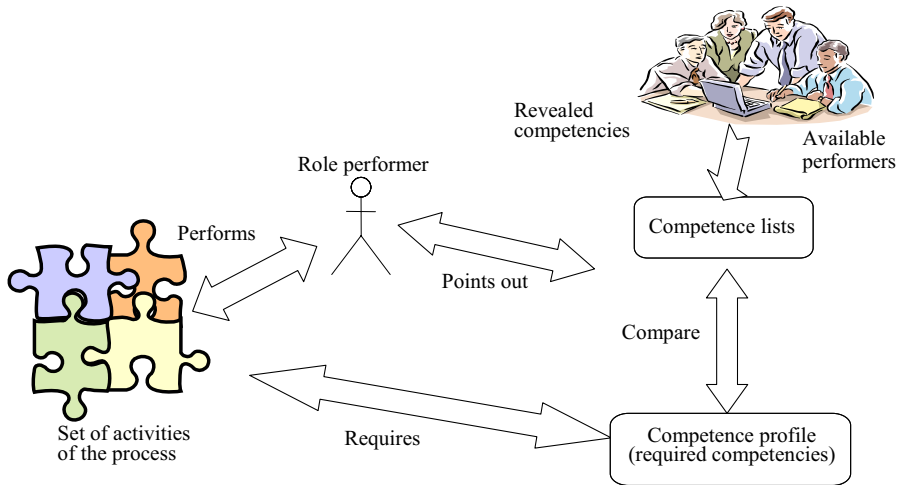


Figure 7. Competence-based selection of the role performer

No software developer is capable and skilled enough to act various required roles. The **competency-based selection** [25], applied in management, may be adopted in software engineering to choose a proper person to play the required role. The term of *competence* means a set of someone's capabilities, features of personality, professional skills, experiences, behaviours and responsibilities. Although the personality type [16] of a given person does not change, the competencies are being developed during his/her professional carrier. Especially, when managers pay sufficient attention to the growth in personnel's competencies.

So far competencies are tested mainly in the recruitment process. It is desirable to maintain and update results gathered during the recruitment – the **competency list** of each employee becomes an important part of his/her personal data. There is a need to work out a current **competency profile** for each required role in the software process. Then competency lists and personal profiles are used to build teams and appoint their leaders, as shown in the Figure 7. Such approach corresponds with the third level of the P-CMM model [12]. The main difference between the P-CMM and the UID approach is that in the first one the competencies refer to internal roles in a company, where in the UID a special care is related to the competencies, and those valuable capabilities, which are required to cooperate and communicate with users.

The level of required professional skills of the suppliers' and purchasers' representatives remains an important problem. The high level of technical skills seem to be not enough to fulfil requirements and realize the pinned hopes. The already applied in other branches competency management seems to be the proper way to improve the software process, especially that following the UID approach.

A personnel of a software company becomes more and more international. In the author's opinion, the root culture and the chosen organisational culture have a great impact on software production. The IBM's experience shows that not all managerial practices may be adopted and accepted in the global scale. Cultural dimensions [15] bear of accepted values. For example, Scandinavian software companies, which prefer female rather than the male organisational culture [15], obtain promising results in software production – the high percentage of successful software products.

6. Final Remarks

There is a long way to improve software process – from identifying symptoms of its ‘disease’, through diagnosis, to the proper therapy and successful results. Direct and indirect users’ dissatisfaction from software products are *symptoms* that software process needs the serious therapy. Technical innovations are not enough to satisfy users although the proven practices and tools should be still applied.

The author’s *diagnosis* is that the isolation of users in the software process causes stated problems. Usually a lot of people are involved by the software product. Unfortunately, in software engineering users stay still separated from developers. The conclusion is that the today’s approach and separation should be abandoned. This conclusion corresponds with the Agile Manifesto¹ – its sixth principle says that *the most efficient and effective method of conveying information to and within a development team is face-to-face-conversation*.

The recommended *therapy* is the UID approach. To sum up, its novelty insists in the briefly remained features: the level of user’s satisfaction from a product is a basic measure of software quality, users’ involvement concerns the whole process (indirect users are represented, too), users’ continual involvement makes a process resistant to constantly changed requirements, an initial survey helps to find out all potential users, users and developers are described by their personality types and various competencies, best practices of hard and soft methodologies retain in use, user-developer relationships are an integral part of the maturity assessment of a software process. The lack of never-ending corrections of software and the high level of its user’s satisfaction should compensate developers for the cost of this approach.

The measure of the *successful result* is the growing level of direct and indirect users’ satisfaction from software products. Most of criteria and implied metrics of a level of users’ satisfaction derive from the software quality characteristics recommended in the ISO 9126. According to the particular product also the other metrics are specified [4], [5], including the time of service in particular cases (from applying for something to receiving the correct answer), the level of facilities available at work, the percentage of real cases supported at work by the delivered system, social atmosphere and comfort of work (as the quality of life), etc.

The emphasis on user’s satisfaction from a product requires different competencies of software developers than it’s been considered so far. Even knowledge of soft methodologies is not enough. The ability to cooperate with users, communication, business vision, ability to imagine functioning of an organization, an empathy for others – are the desired competencies, among others. New competencies require, in turn, changes in education at the university level [7]. Those competencies should be taken into considerations also in the recruitment process for studies and for a job. Some developers are preferred to work only in the back office rather than in the front office – in analogy to solutions applied in banking.

User-oriented methodologies cannot depend on someone’s good will only. In the author’s opinion, the maturity level of the software process should depend also on user-developer relationships. The author’s aim and research is to expand the CMM model. The proposal of the UR-CMM (*User Relationship Capability Maturity Model*), in analogy to the People CMM model, is under development.

¹ Principles behind the Agile Manifesto, <http://agilemanifesto.org/principles.html>

So far, software companies are hardly interested to cooperate with users' representatives. One example comes from the IBM – keeping the focus on what is being produced is an essential feature of the *engagement model* of the process [10], which consists of four components: Work Product Description (over 300 WPDs are in use), Work Breakdown Structures (WBD), a set of required roles (described by sets of skills), and a set of applied techniques. This approach and all specified roles are limited just to the producer's side.

Agile approaches as the opposite to rigid ones are supposed to grow gradually as the users see that products of hard methodologies seldom satisfy them. The described UID approach can help to find a successful solution.

References

- [1] Arbaoui S., Lonchamp J., Montangero C., The Human Dimension of the Software Process, Chapter 7 in: David Graham Wastell (ed.): *Software Process: Principles, Methodology, Technology* (1999), pp. 165–200
- [2] Beck K., *Extreme Programming Explained*, Upper Saddle River, NJ, Addison-Wesley, 2000.
- [3] Begier B., Quality of Goals – A Key to the Human-Oriented Technology, *Australian Journal of Information Systems*, vol. 9, Number 2, May 2002, pp. 148–154
- [4] Begier B., Software quality assessment by users (in Polish: Ocena jakości wyrobu programowego przez użytkowników), Chapter 27 of the monograph on *Problems and methods of software engineering*, WNT 2003, pp. 417–431
- [5] Begier B., Wdowicki J., Quality assessment of software applied in engineering by its users (in Polish: Ocena jakości oprogramowania inżynierskiego przez użytkowników), 4th *National Conference on Methods and Computer Systems in Research and Engineering, MSK'03*, Kraków, 26-28 listopada 2003, s. 547–552.
- [6] Begier B., Customer relationship management in software company (in Polish: Zarządzanie relacjami z klientem w firmie informatycznej), in: *Software engineering. New challenges*, WNT, Warszawa 2004, pp. 213–226
- [7] Begier B., Suggestions concerning reorientation of software developers to focus on a broad spectrum of users, in: *Proceedings of the ETHICOMP 2005*, Linköping, Sweden, September 22-15, 2005.
- [8] Boehm B., Egyed A., Kwan J., Port D., Shah A., Madachy R., Using the WinWin Spiral Model: A Case Study, *IEEE Computer*, vol. 31, July 1998, pp. 33–44
- [9] The challenges of complex IT products. The report of a working group from The Royal Academy of Engineering and the British Computer Society, April 2004, www.bcs.org/BCS/News/positionsandresponses/positions
- [10] Cameron J., Configurable Development Processes. Keeping the Focus on What is Being Produced, *Communications of the ACM*, vol. 45, No. 3, March 2002, pp. 72–77
- [11] Checkland P., *Soft Systems Methodology in Action*, John Wiley & Sons, July 1999
- [12] Curtis B., Hefley B., Miller S., *The People Capability Maturity Model*, Addison-Wesley, Boston (2001)
- [13] DeMarco T., Lister T., *Peopleware: Productive Projects and Teams*, Dorset House, New York (1999)
- [14] Grudin J., Pruitt J., Personas, Participatory Design and Product Development: An Infrastructure for Engagement, *Proc. of the Participatory Design Conference 2002*, pp. 144-161, accessed in May 2005 at: <http://research.microsoft.com/research/coet/Grudin/Personas/Grudin-Pruitt.pdf>
- [15] Hofstede G., *Geert Hofstede™ Cultural Dimensions*, <http://www.geert-hofstede.com/>
- [16] *Information About Personality Types*, <http://www.personalitypage.com/info.html>
- [17] zur Muehlen M., *Business Process Automation – Trends and Issues*, Center of Excellence in Business Process Innovation, Wesley J. Howe School of Technology Management, Stevens Institute of Technology, Hoboken, New Jersey, USA 2004
- [18] Pressman R., *Praktyczne podejście do inżynierii oprogramowania* (in origin: *Software Engineering: A Practitioner's Approach*, New York, McGraw-Hill Companies 2001), Warszawa, WNT 2004
- [19] Rajlich V.T., Bennet K.H., A Staged Model for the Software Life cycle, *IEEE Computer*, vol. 33, July 2000, pp. 66–71
- [20] Scheer A.-W., *ARIS – Business Process Modeling*, 3rd edition, Berlin – Heidelberg, Springer Verlag (2000)
- [21] Stoner J.A.F., Freeman R.E., Gilbert D.R., *Management*, Prentice Hall Inc., (Polish version: *Kierowanie*, 2nd ed., Warszawa, PWE (1999)
- [22] Turner R., Boehm B., People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods, *Cross Talk. The Journal of Defense Software Development*, December 2003, pp. 4–8
- [23] Wieger K.E., What Makes Software Process Improvement So Hard?, *Software Development*, February 1999, on-line edition: http://www.processimpact.com/articles/spi_so_hard.html
- [24] Winograd T., *Bringing Design to Software*, Profile 14. Participatory Design, Addison-Wesley (1996).
- [25] Wood R., Payne T., Competency-based Recruitment and Selection, Chichester (UK), John Wiley & Sons (1998).

Agile Software Development at Sabre Holdings

Marek BUKOWY^a, Larry WILDER^b, Susan FINCH^b and David NUNN^c

^a *Sabre Polska, sp. z o.o.; ul. Wadowicka 6D, 30-415 Krakow, Poland*

^b *Sabre Inc., 3150 Sabre Drive, Southlake, TX 76092, USA*

e-mails: {Marek.Bukowy, Larry.Wilder, Susan.Finch}@sabre-holdings.com
<http://www.sabre-holdings.com/>

^c *Valtech, on assignment at Sabre Inc., Southlake, TX, USA*

e-mail: David.Nunn@valtech.com

Abstract. A transition to an agile software development methodology in a heavily waterfall-oriented organization is discussed. A collection of agile software development practices implemented at Sabre Holdings is described. A summary of the desired set of practices is given, along with data summarizing the degree of pervasiveness of these practices in real life. Quantitative results on the use of the methodology in day-to-day development are presented. Lessons learned and mitigation ideas are also discussed. The gathered information shows that the habits of the old model can strongly encumber the implementation of an agile methodology, but even implementation of a few practices yields very tangible results.

1. Who is Sabre?

Sabre Holdings is a world leader in travel commerce and the travel industry: from merchandising and distributing travel by booking and selling air, car, hotel, cruise and other types of inventory, through online and offline points of sale, to providing a complete software portfolio for hundreds of airlines, to supplying technology solutions across the industry. We serve travelers, corporations, travel agents and travel suppliers around the globe. The Sabre Holdings companies include:

- Travelocity, the most popular online travel service (www.travelocity.com)
- Sabre Travel Network, the world's largest global distribution system (GDS), connecting travel agents and travel suppliers with travelers; and
- Sabre Airline Solutions, the leading provider of decision-support tools, reservations systems and consulting services for airlines.
- In addition to company-specific development organizations, a pan-Sabre Holdings software development organization exists. This is the organization in which the transition described in this paper took place.
- From the software development point of view, Sabre features:
- High profile customers demanding ultra-high quality of products for their business-critical applications,
- Long customer relationships resulting in multi-project programs,

- Multitude of environments, from legacy/mainframe to midrange server farms to desktop applications, interacting with a variety of sources of data in real time, utilizing a mixture of programming languages and frameworks,
- At times, extreme complexity of business logic reflecting the underlying rules and procedures of the airline business.
- Very competitive market following U.S. and European markets deregulation promoting rapid requirement changes and short development timelines.
- Geographically distributed teams, with distances anywhere from one room to multiple buildings, cities or continents within the same team.

History of software development at Sabre Holdings dates back to 1960's and mainframe applications. This comes with a strong bias of processes towards what was the best methodology applied all over the world during the time when mainframes were mostly used – which was waterfall.

2. The Agile Program

Sabre Holdings' experience with waterfall was no different than most of the industry.

Requirements were often unclear, sometimes contradictory and always changing. Some systems were extremely complex, and gathering all the requirements at the beginning of the project was more difficult than designing and implementing the corresponding system. Increasing the rigidity of the process proved not to be the answer – it slowed software construction and created order-taking behaviors.

The Product Development organization looked from 2001 for an agile methodology that could be implemented with little controversy and a high level of adoption.

We liked the values promoted by the Agile Alliance in its *Manifesto for Agile Software Development*, summarizing what takes precedence in an agile methodology:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

We wanted to change the organization in the spirit of Lean Software Development. However, we didn't find a good fit within the agile methodologies existing at the time. Most were declared to support small teams, up to 10 people. Some of our teams were much bigger due to complexity of our business. Most methodologies had no success track record for distributed development – while our projects required expertise located in multiple sites, from 2 to 4 in each case, on at least 2 or 3 continents. Many methodologies targeted just the software construction activity – rather than the whole lifecycle. We needed something to replace Waterfall at its full scope and we tasked Valtech with defining a methodology that would help us.

The closest to these requirements was the Rational Unified Process, however it was not lightweight (or “lean”) enough. A number of practices from more agile methodologies like Scrum were combined with the RUP's framework and the resulting mix was named Agile UP, or AUP for short.

It's important to realize that AUP had to plug into the existing system of waterfall-specific PM processes. Most changes are therefore seen on the lower levels such as development teams, and less – on the levels of aggregated project management. The descriptions below discuss the ideal target shape of AUP, the real-life accounts are given in the following section.

2.1. Actors

The attempt to define special roles within AUP teams, such as iteration tracker, weren't very successful. All other roles had their waterfall-process counterparts and regardless of the name (e.g. Business Analyst vs. Consumer Advocate) performed the same functions. The key thing was that the way to do it had now changed.

2.2. Timeline

Development in AUP is iterative and each phase consists of one or more iterations, usually 2 or 4 weeks in duration. The development is organized in phases, similar to the 4 UP phases, each with specific objectives:

Phase	Objective	Ends when
Inception	To verify that the project is worth doing and to reach agreement among all stakeholders on the objectives of the project	A good idea of the functionality has been reached and preliminary budget has been estimated and assigned
Elaboration	Reduce project risk and to stabilize the requirements and architecture.	Development effort and operating costs can be estimated
Construction	To incrementally develop a complete product through all construction phase iterations.	Stakeholders decide that the functionality is sufficient for the customer
Transition	To deliver the product to the user community.	Customer is running the last release in production

The length of phases and their proportions depend on the project characteristics and the team's ability to achieve the objectives of the phase. For example, the length of Elaboration will be longer for more complex projects, where a new architecture is required and therefore a final estimate is more difficult to give.

2.3. Planning

Work items are based on use cases (user-perspective functionality), engineering cases (non-functional technical requirements, such as operability, security etc.) and risk mitigation (action points from risk analysis). The items are identified, estimated and added to the *backlog* of deliverables. They are changed, added and removed during the project as new information comes.

Every deliverable is named in terms of demonstration, implying acceptance criteria (e.g. "demonstrate that category 4 validation is performed on one-way one segment

itineraries” rather than “category 4”). The description has to be unambiguous enough for the work to be estimated before the iteration, it also makes it testable. A deliverable can only be accepted if the effort estimate for it fits within an iteration, otherwise it has to be broken down into smaller pieces.

Backlog is worked in value order, based on risk mitigation and business value. All the high risk items are to be resolved during elaboration. The construction is about “doing the obvious” and is prioritized according to business value – in order to maximize customer value for the software released periodically during construction. Dependencies are resolved dynamically when choosing the next achievable deliverable.

Every iteration ends with a demonstration of compliance of work items with the acceptance criteria. Like in other iterative methodologies, items that haven’t been fully completed in one iteration are not to be counted and have to slip to the next one.

The analysis, design, coding and testing activities are performed for every deliverable separately. The analysis step is to be done as late as reasonable. More complex analysis has to start sooner and be completed just before the iteration in which the deliverable is to be worked by the development team. The testing can begin right then – at least in the sense of defining the test cases. The design of the deliverable, coding of the tests and actual programming of the application code make up the scope of the deliverable work.

2.4. Tracking

Day to day progress is tracked using stand-up meetings, where team members report their progress, obstacles, plans and lessons learned. This is a big enabler for an effective exchange of information – it is estimated that every participant should get a maximum of 2 minutes. The issues raised can be followed up on after the meeting.

The overall project progress is tracked as part of the iteration planning for the next iteration(s). This allows feeding back the budget and time estimates for the *whole project* (including the undone part) to the customer, effectively allowing the customer to react early and change direction so that the project brings the expected value.

A number of variables are tracked and combined to provide a project level burndown information. The tracking points are only those at the end of every iteration; unlike Scrum, AUP does not track “sprint burndown” – during the iteration – for the fact that the effort required hasn’t been found justifiable by the additional information it would bring.

2.5. Feedback

Getting early and regular feedback is the foundation of the methodology. At the end of every iteration, information is captured and analyzed, allowing re-planning the remaining part of the project.

Every iteration ends with a demonstration which only occasionally would involve a customer. Very often, a customer representative, or “proxy” would be present instead. This would be the consumer advocate / definer, usually affiliated with the marketing department. One of the reasons for this choice over a direct customer consultation is the fact that many products are constructed to serve a few customers. Rather than having different customers with slightly different ideas for the same product, who perhaps at times shouldn’t even know about each other, in the same room, marketing would

communicate with them “asynchronously” and align the requirements to be acceptable by all prior to involving the developers.

Another type of demonstration is the one involving the real customer. This typically happens much less frequently, only a few times during development, the last of which is at the official customer acceptance phase.

Feedback is not limited to the product. The whole team meets and discusses practices and how the way they work should be changed for the next iterations. This results in so-called plus/delta notes (plus=what we want to do more of, delta=what we want to do differently).

2.6. Ceremony

Many waterfall documents went away. Some of the formerly required document-based milestones could still be used in agile projects, subject to a subtle redefinition of their role in the process.

10 types of artifacts are defined for AUP: Vision, Business Case, Use-Case Model, Supplementary Specification, Risk Assessment, Backlog, Estimate, Master Test Plan, Phase Plan, Software Architecture Document. The Software Architecture Document is very important, as it must be created for every project and analyzed by the Sabre Architecture Review Board prior to approving a project, regardless of the methodology used. The SARB oversees all projects and formally enforces technical alignment of all projects, in the areas ranging from the high level flow and architecture to technologies used. As we will see, many of the artifacts refer to or are based on SARB documentation.

As a matter of principle, in order to avoid creating unnecessary documentation, the direction in requesting documentation is shifted from the “push” model – one where it is mandated by the process and created with little motivation, to a “pull” model – one where anyone needing information looks for the source of information and drives putting it in a form suitable (or rather – sufficient) for their purposes. This requires maturity and self-management of the teams.

2.7. Engineering

There are really only a few guidelines that are defined here. All of them are known from other agile methodologies.

- Code should be kept in a source code repository; checked out only briefly, worked, checked in, and frequently re-built with other recent work.
- Every project must have a Continuous Integration site. The CI site should be able to perform automatic builds and tests upon every check-in of code into source control.
- Automated unit tests must be built while (or before) coding the covered functionality.
- A regression suite has to be maintained to ensure that changes to the code base aren’t damaging the functionality coded earlier.
- Orientation manual should be prepared for new team members to allow them to quickly configure the environment and understand the basics of the project.

- Code reviews are to be individually agreed between developers and performed prior to demonstration.
- Code refactoring is encouraged but cannot be mandated.

3. Adoption

The adoption was driven by a small “Catalyst” group consisting of 2 subgroups:

- Project management – 3 Sabre project managers and 2 Valtech consultants, focused on coaching the project management community to better understand the principles of agile practices and prepare for the transition. They selected the projects and teams for each AUP project, explained the methodology and helped kick-start the backlog. 3 out of 4 authors of this article served on this team.
- Engineering – 1 Sabre developer and 1 Valtech consultant. This subgroup provided coaching and propagated the engineering-specific practices among developers during the first 2-3 iterations of every project. Involvement in no more than two projects at a time.

Since the largest team and variety of actors were located in the company’s headquarters, the Catalysts operated mainly there. Other sites, featuring mostly developers, followed with the level of adoption of practices determined by their eagerness to match or exceed the example set by developers at the headquarters.

Some of the projects involved were already running and their environments had to be redesigned for AUP. The results of implementing AUP over an existing code base depended largely on the clarity and modularity of the code, as the amount of time that could be spent on re-architecting the environment was usually low.

The degree of practice adoption varied greatly between all participating actors.

3.1. Customers and Analysts

The first projects to adopt AUP were internal. Thus, all risks associated with introducing the new culture were covered by the company. Lessons learned by the early adopters helped mature AUP before applying it to external projects.

The key to enabling introduction of an agile methodology at Sabre for external projects were the customer relationships. Well known in the travel industry, with its relationships with airline or agency customers dating back decades, Sabre was in a position to convince a few customers to run some projects in an agile way, especially if these projects represented a fraction of the development done for the customers.

Most of the external contracts, while roughly declaring the time and budget for the projects, were mainly constructed on a Time and Material basis. This, combined with the ability of specifying the details later, provided the flexibility that AUP needed while ensuring the control that the customers wanted to keep.

The customers had to switch their mentality and learn how to play their part of the methodology effectively. The learning would last anywhere from one to three or more projects depending on a customer. At first, the risk would be taken internally by developing seemingly waterfall-style, fixed scope projects with a vague definition at

the start. In that mode of operation, the customers would be regularly asked during development to increase the level of detail of requirements by using examples, use cases etc.

The customers would also be invited to review the deliverables and provide feedback. Getting them to regularly participate was a challenge at first. However, they quickly realized how beneficial this was to the projects and learned the value of feedback. Thanks to that, quite a few significant miscommunications of requirements could be detected early in the process, saving everyone's time.

The next step was to encourage the customers to abstract all requirements and specify them in terms of the deliverable definition. In the best cases, we were able to define a backlog sufficient for starting work within 3-5 days of intensive sessions.

Thanks to the simplicity of specification of the requirements by test cases, it was sometimes possible to take it to the extreme of test-driven development by having business analysts specify functionality via XML files with request-response pairs.

3.2. Management

Tied back to the "safe", because well-known, haven of waterfall methodology, many managers and project managers struggled with the notion of self-managed teams. Elaborate and detailed iteration plans covering the whole project were often built before the first line of code was written, and then sequentially "executed".

A distinction must be made here between two project roles that might be specific to Sabre. A *delivery manager* is a relatively technical role with which the responsibility for the delivery of a project resides. S/he ensures that the team can produce their deliverables. A *project manager* is responsible for project tracking and administration.

Delivery managers, focused on the shape of the product being implemented, very quickly appreciated the early feedback provided by the new methodology and used the information to drive the projects more optimally from the technical standpoint.

There were more challenges with project managers' tracking. They often complained that AUP did not give them and the executives as many metrics allowing insight in the progress of the projects as waterfall did. Certainly, the set of metrics was small, different and no "automatic" translation could be performed. Some of the metrics weren't integrated in the same project tracking tools, a gap which could be bridged only by those with good understanding of principles of AUP.

It was particularly hard to alter the project management practices that were enforced by previous project management standards. For example, in AUP, partially completed deliverables are considered 0% done, which is in clear conflict with the PMI standards for project tracking. Agile methodologies have not been around long enough to gather an equal amount of respect and therefore such conflicts would, by default, invoke PMI behaviors until eradicated by systematic mentoring.

Focused on fulfilling the indicators of "agility", many PMs would promote "catching up" on unit test coverage by requesting the developers to add unit tests long after the corresponding functionality or, worse, complete program modules were coded. While not completely pointless, as improving the code's future maintainability, this wasn't nearly as beneficial as it could have been.

A number of approaches were tried for estimating the effort of the deliverables. All revolved around some forms of a planning poker, where developers would repeat their estimates on every item until they all agreed. A few options were tried, resulting in

a similar inaccuracy. The fastest results were obtained by reducing the effort to the minimum and deciding whether an item will *fit* in an iteration.

In some teams, estimates would be decided at the technical lead level, resulting in a similar inaccuracy but much greater level of frustration of the team.

3.3. Development Teams

The level of acceptance varied greatly for the many practices that the developers were supposed to adopt. Very few projects implemented all of them in the ideal shape known from literature.

Stand-up meetings – there were two difficulties with introducing stand-up meetings. First, developers had a hard time admitting failures/obstacles. They were gradually becoming more open as the teams worked together longer and saw that hiding a problem was worse than admitting it. Some teams have grasped the idea faster than others, realizing that early problem identification enables quick progress.

Secondly, it was very challenging to keep the meetings short. Used to longer, usually less frequent, design/analysis meetings, teams would tend to take 30-45-60 minutes, turning a quick report meeting into a collective issue resolution. Taking issues “offline” was a challenge particularly for distributed teams, where the overhead of organizing another meeting was greater, so the issues had to be narrowed down during the main meeting enough to determine at least who should be contacted for resolution of the problem outside of the meeting.

For large teams, working on a variety of areas within the project, it became possible to order the speakers by area and have them join for their part only, while a handful of people supervising the progress would take part in the whole meeting.

Code reviews – as those were not explicitly required, very small number of developers have actually requested their peers to review their code.

Continuous integration – the level of adoption was mainly related to the complexity of the software environment. Some projects with giant build times had the builds done as infrequently as once a day, others had their builds trigger automatically upon every check-in of new code under source control.

Unit testing – a wide diversity was observed here, too. In case of most projects that were requested to “go agile” months or years after they started, the developers felt that the code they worked with was not modular enough to allow unit testing.

The results were much better for projects done in AUP from scratch. Developers learned to design their code in a way enabling “plugging in” regression tests on different levels. There were always developers who resisted, but a combination of determination of a lead and a role model on the team, who knew how to get the best value out of testing, helped greatly in achieving reasonable code coverage, up to 70%.

It was crucial to keep the coverage high at all times. Developers rightly pointed out that adding unit tests after the fact did not speed up development and required additional time which was not scheduled. Enforcing high coverage after coding of the corresponding part resulted in trivial test cases at best. Manual functional testing was done in all projects with intensity inverse to the code coverage of automated tests.

Backlog – initially, personal pride would not let the developers take opportunity of the flexibility of the backlog. Most would work overtime to finish all assigned deliverables even if they cut down on the above practices or quality of their code. After some coaching, teams learned to avoid over-committing as they realized the consequences could drive overtime, bad quality and reduced overall productivity.

Geographical distribution – it has to be stressed that contrary to certain practice-oriented guides to agile development, the geographical distribution of team members has not been a show-stopper. Embracing certain simple rules for communication and adjusting schedule to increase availability for discussion during time windows was key. Using written communication and having time-zone differences showed its advantages. Email and instant messenger discussions ensured that a good amount of information was captured in writing and could be referred to in the future. Conference calls and ad-hoc phone calls provided enough interactivity to allow the teams to progress.

Paperwork – the official documentation was limited to the necessary minimum. Only the most stable facts were captured in writing, others had to remain fluid. Even some of the AUP-defined documents were identified as waste:

Artifact	Description	General attitude
Vision	Defines the problem being solved, stakeholders, and features of the project.	OK not to have it
Business Case	Provides the business justification for the project	Usually vague
Use-Case Model	Use Driven Model of the Functional Requirements of the system	Depending on the type of project. Almost always present for GUIs, but maintained rather as an old-style functional specification for business logic oriented backend applications.
Supplementary Specification	Defines the Non-Functional requirements of the system	Most often in form of action points from an architectural review; translated straight into non-functional deliverables
Risk Assessment	Identifies project risk and mitigation	Rarely as a document – rather a series of action points
The Backlog	Lists all deliverables of the project	A living spreadsheet
The Estimate	Provides the cost of delivering the project	Simple ones worked just as fine
Master Test Plan	Defines how the system will be tested, i.e. the test strategy	Always a challenge to get a complete one
Phase Plan	Lists the number and length of iterations, phase begin/end dates, major milestones, etc.	Project plan often maintained in the spirit of waterfall – tasks/dates
Software Architecture Document	Describes the software architecture of the system	A very high level presentation fulfilling the requirements of the old process to get approval for the project

The low levels of documentation sometimes surprised, but it was invariably at the teams' discretion to find the right balance.

3.4. Cultural Challenges

The most incompatibilities of the corporate culture with the agile process were displayed on the level of behaviors and habits. Having been a large airline's data processing center in the past, we still occasionally see shadows of strict command-and-control type leaders from the military beginnings of the airline industry. The undesirable behaviors identified include "learned helplessness", disjoint silos of knowledge and responsibility, "hand-off" culture (as opposed to end-to-end collaboration) that have grown over the years in the heavily process-driven environment. AUP, on the other hand, required behaviors like open collaboration within wide cross-area teams, feeling of accountability against peers rather than a supervisor, initiative and ownership stimulated by stewardship delegation, ability to think outside of one's comfort zone, broad view of the project/systems. In fact, a significant effort that started around the same time as AUP deployment, has been promoting the necessary values and has laid the foundations for self-managing teams.

Although these attributes are difficult to measure, their effect on the quality of the code produced is remarkable. Many issues that normally would not have been discovered until customer testing are identified much earlier, sometimes within the first few weeks of the project. There's visibly less rework, significant changes of direction or unused features that would have otherwise been requested "just in case" at the definition phase.

There simply are no major surprises at the conclusion of an AUP project. Because the customers are involved from the beginning and see progress through code demonstrations each iteration, misunderstandings and miscommunication are discovered and corrected quickly.

4. Results

Despite the challenges of making the transition, we have seen significant improvements in product quality. A good base for comparison is a pair of projects from the same domain, but differing in scope: V1 – executed a few years ago in waterfall, and V2 – carried out more recently in AUP (see Figures 1 and 2).

V1 consisted of 2 main areas; A and B. V2 is focused mainly on area A, similar in concept to that of V1, but functionally much broader – the complexity of A in V2 is about 3 times higher than in V1. V2 had to be created almost from scratch, using less than 5% of original V1 code. The last figures here come from March 2005, when V2 was at the end of its construction phase, counted 98,000 lines of code for area A and test code coverage was at 31%.

The pattern of defects in V2 is drastically different to that of V1. The total number of high severity bugs detected during system testing has been reduced by a factor of 10. The figures below depict the number of open defects at different points in time.

V2 shows a significantly lower level of priority 2 (work stoppage) defects than V1 that were plaguing the development. Priority 1 (system failure) errors are practically non-existent in the AUP project, while they were common in V1 and strongly limited the ability to test many program paths.

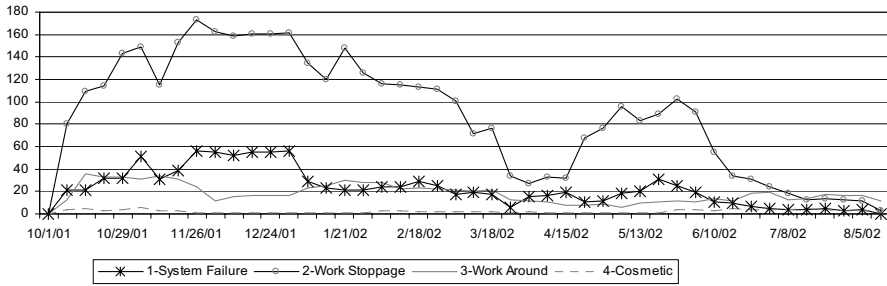


Figure 1. V1 Open Defects

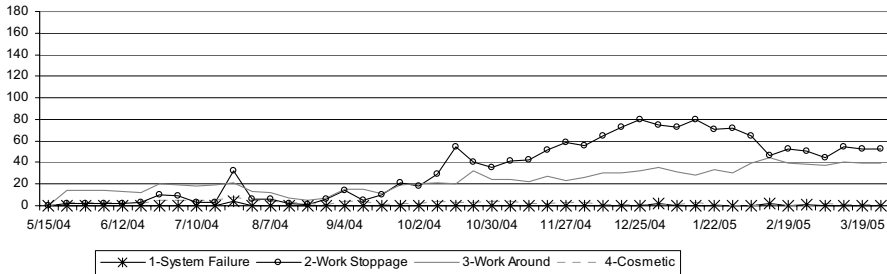


Figure 2. V2 Open Defects

The quality metrics alone may seem insufficient. Many aspects escaped unmeasured – as the set of tracking data was consciously being kept small. Collection of more data than needed to make development progress steadily would be contradictory with the agile philosophy itself. Instead, subjective feedback was regularly gathered from the teams as well as customers. The dominance of positive feedback was quite convincing for our leadership and led to a decision to stay with AUP and not add ceremony by collecting more data.

The agile approach at Sabre has undoubtedly proven itself and the number of projects led in accordance with its principles is constantly growing. The implementation of AUP is far from complete, but even though some practices are not fulfilled very well yet, we are observing a massive shift towards the Agile Manifesto values.

5. Lessons Learned

A few closing remarks can be made on what we have learned during the process so far. Hopefully they can help others to avoid traps of moving towards an agile methodology.

- It is vital to thoroughly explain the principles behind practices.
- Overlaying agile practices over an old culture might result in an iterative waterfall.
- Degree of adoption is proportional to the effort put in mentoring the teams.
- Switching to AUP during the project requires reengineering of the code.
- Iteration is the main catalytic factor for getting regular feedback.

- Mandating code coverage does not guarantee quality.
- Measuring practices blindly does not work – what works is focusing on quality and giving the teams freedom to figure out how to achieve it.

6. Conclusion

The migration from traditional waterfall development to AUP has seen significant challenges. The degree to which waterfall thinking influences the behavior of development teams cannot be overstated. The implementation of an agile development model must include the effort necessary to develop the appropriate skills for each role in the development team as well as to overcome the cultural challenges.

Defining practices and procedures appropriate to agile and teaching teams those definitions will be insufficient to make the transition. Teams will simply transpose those definitions to fit within their waterfall model. Only the names teams use to describe their work will change, however the underlying behaviors of the team will still follow the waterfall model.

To effectively make the transition one must identify those within the organization, or externally if internal resources do not exist, who understand agile philosophy and use them as coaches and mentors with project teams. Deep understanding of agile concepts comes mainly through experiencing them on real projects.

Allowing flexibility and introducing feedback into the process are the two single most important factors for success. Introduction of individual practices is not as critical as long as the teams are accountable and flexible to find the best balance.

The ability to increase the feedback and flexibility depends on customer openness and willingness to cooperate. Sabre has great customers who recognized the value of moving towards a collaboration setting. Changing the relationship with the customer is a *sine qua non* condition for fully implementing any agile methodology.

No elaborate process, even one composed of agile practices, but executed blindly, can substitute human responsibility and other behaviors that determine the success or failure of an organization.

References

- [1] Agile Alliance web site, <http://www.agilealliance.org>
- [2] Cockburn A., Agile Software Development, *Addison-Wesley Professional*, 2001
- [3] Poppendieck M., Poppendieck T., Lean Software Development: An Agile Toolkit, *Addison-Wesley Professional*, 2003
- [4] Larman, C., Agile and Iterative Development: A Manager's Guide, *Book News, Inc.*, 2004
- [5] Larman, C., Basili V., Iterative and Incremental Development: A Brief History, *IEEE Computer*, June 2003, <http://www2.umassd.edu/SWPI/xp/articles/r6047.pdf>
- [6] Szulewski, P. and Maibor D., What's New and Some Real Lessons Learned, 1996; <http://www.stsc.hill.af.mil/crosstalk/1996/03/LessLear.asp>

Workflow Management System in Software Production & Maintenance

Paweł MARKOWSKI

ComArch S.A.

e-mail: Pawel.Markowski@comarch.com

Abstract. The maintenance is the longest period of software product life cycle and the most expensive, therefore increasing efficiency and reducing costs in this domain is important issue for any software company. Treating maintenance and also product development as business processes, we can use workflow software to optimize and automate their execution. Tytan Workflow is advanced system for business process modeling which is used in software engineering related activities. In this article features of system and examples of process models was introduced.

Introduction

Nowadays for software company, process of creating IT system is not only its production, sale and implementation at client's side. Also very important part of this process is what is going on with product afterwards. Software maintenance is the last stage of system life, during which IT firm has to care about software proper work and its development – stage different than others, because product is running in daily business activity. This stage also distinguish itself in terms of costs. Various sources have consistently shown that it consumes up to 90% of the total life cycle costs of software.

The main reason of this state is long period of time in which the system has to be supported. Software is often used far longer than primarily it was planned (good example how challenging could it be was Y2K problem) and maintenance phase usually lasts for years. And all this time the system resides in constantly changing real-world environment, which forces changes in software and its evolution in order to better fulfill goals for which it was created.

Software maintenance activity has been classified into four types [3]:

- Perfective: responding to user-defined changes resulting in system enhancements
- Adaptive: changing the software to adapt to environment changes
- Corrective: correcting failures, fixing bugs
- Preventive: performed for the purpose of preventing problems before they occur

Each of these elements requires primarily deep analysis and understanding of currently working system, what usually is quite laborious (and therefore expensive). The main reason of this situation is that maintenance of software is not performed by persons who created the product. Besides, process of understanding working software

is also affected by the environment in which product is running, skills and experience of members of support team, changes which were made in the system during hitherto process of maintenance and also time available for application of change.

There are no simple solutions in that case. Partial improvement may be obtained by implementation of ISO quality standards, which could systematize work. Helpful would be simply increase in efficiency of activities which are parts of the software maintenance process, through their automation.

1. Work Flow Management in Software Engineering

Assuming (which is the fact indeed) that processes related to software maintenance are for software company only ordinary business processes in which we are dealing with different groups of people responsible for separate activities areas, exchanging between each others informations and documents, we can try to automate work flow management in order to increase its efficiency and decrease costs. Therefore in this domain without fail could be used software for Workflow Management (called also WFM).

These tools are automation of business process, whole or only one part, in which elements of the work are transfered to the following participants in order to execute given actions, according to procedurals rules of process functioning. Owing to WorkFlow, activities forming business process can be allocated, transfered, activated and managed basing on the rules reflecting business operations and decision process. Here are introduced relations between three activity domains in company:

- business rules that create process mileage
- organization structure of company – persons related with particular stages of processes
- information which is exchanged between various process stages

2. Tytan Workflow

Solution used in ComArch S.A. to manage work flow in the company is application Tytan Workflow. In order to fulfill company needs, applications (using Tytan platform) were implemented to serve such processes like bug solving, change request management or management of creating project process. Tytan Workflow is a modern tool that enables automation of business processes allowing documents, information and tasks to be routed between users, according to a set of well defined rules. It is developed according to standards and recommendations produced by the Workflow Management Coalition (<http://www.wfmc.org>).

2.1. Architecture

System is composed of three elements [7]:

- **web server layer** – responsible for analyzing client requests, delegating them via IIOP to the application server, collecting intermediate results in form of

XML and sending the results in form of HTML pages. Used technology is Jakarta Tomcat.

- **application server layer** – responsible for web server requests serving. Included in the container EJB components are responsible for application business logic. The communication between EJB components and database is done via JDBC protocol. This layer serves different services used to manage the system and primarily is running engine which is processing instances of business process definitions. Used technology is Jboss.
- **database layer** – The basic element of the TYTAN system is the uniform database containing complete files. This database stores also all user data, processes, tasks and documents from the administration records and configuration units. Used database is Oracle.

2.2. Process Model

The basic Tytan Workflow functionality is defining and modeling business processes. The reference model defines a business process [2] as *a procedure where documents, information or tasks are passed between participants according to defined sets of rules to achieve, or contribute to, an overall business goal*. A work flow is a representation of the business process in a format understandable for given application. A work flow model is an acyclic directed graph, in which nodes represent steps of execution and edges represent the flow of control and data among the different steps. Components describes below are used in Tytan Workflow to creating workflow model:

1. **Process:** This is so called definition of process, which consists process name, version number and start conditions. It describes from what steps, called activities the process is made, what is the order in which activities are executed, what are groups of persons, that are responsible for particular parts of process. It could be said that process definition is build from activities connected with transitions.
2. **Activity:** This is single step of process. In Tytan Workflow there are two types of activities:
 - automatic – are used for calculations, external, system calls, database operations and even printing
 - manual – dispatched according to activity allocation algorithms selected from the algorithms available in the system (e.g. first available worker, load balancing).

Besides we can divide activities by their functions – there are beginning activities, finish activities, subprocess activity (it starts other process), decision activity (node in which one of possible paths is chosen) and few more. Activity definition describes activity name, type, preconditions (which have to be satisfied before activity execution), exit conditions (have to be satisfied before activity ends). It also lists parameters, that will be shown to user and how will they be displayed (in what format and order), which parameters will be copied to other process. Definition lists also all operations [see below] which will be called during activity execution – there is possibility to specify in which phase operation will be performed.

3. **Transition:** Connector which indicates possibility of transition from one activity to other. Definition of transition consists starting activity, target activity and condition which has to be satisfied in order to allow process follow this path.
4. **Parameters:** Parameters are data related to process. They are of various types reflecting data types of columns in oracle database which is back-end for Tytan Workflow. During activity definition decision whether particular process parameters will be displayed for user and in what format, whether will be possibility of changing value of given parameter and whether parameter is mandatory, whether parameter will be copied to other process or whether it will be global parameter with common values for few processes.
5. **Operations:** Workflow operations are sets of commands executed by Workflow Engine. They are calling API functions from application server and use parameters as their arguments. Operations makes possible substantial widening of process functionality. They are simply procedures written in one of programming languages supported by Tytan Workflow and executed during step of process. Operations may be used to perform the whole spectrum of actions, difficult to realization with only activities connected with transitions e.g. complicated computations, string operations, processing various data.

2.3. Defining Process

Administration Application

For administrators who want to create new process model in Tytan Workflow, the administration application was created. The application allow to create, edit and remove almost all aspects of process definition. To create new process model it is necessary to build diagram (graphical representation of work flow model) using advanced graphic editor designed for this purpose. User can choose given activity type from list of available activity types, place it in canvas and connect using transitions. Using the graphic editor there is possibility to define all other aspects of process definition:

- start and end conditions in activities
- conditions in transitions
- list of parameters that will be shown in related to given activity html outputs
- parameters properties (is it read-only or editable, what should be format when displaying it, which position in output page should it occupy)
- choice of operation which should be called in given activity and its argument and phase of execution

Lanes

Process cannot be startup in organizational vacuum – to provide users possibility of using process it is necessary to link its definition with groups of persons. It is realized with the aid of entities called lanes. Lane is related with set of given groups or set of group types (or mixed first and second) – this relation can be modified by user. Lanes are defined in process context – there may be many lanes definition in one process definition – this feature provide ability to advance division of responsibility sections within one process. Some group of activities may be allocated to one lane and other group to other. It has to be mentioned that lanes describes only set of potential workers

who can get given activity. Which concrete user will have it, depends on allocate algorithm defined for this activity.

Versions

Process definitions have versions. Newly created definition has open first process version. In any time there is possibility of creation next process version basing on currently edited version. Newly created version will be exact copy of last edited definition, what enables chances of multi-step definition development, through adding some changes in following versions. Given process definition cannot have more than one active version. All new process instances of given definition are created in active version (excluding subprocesses, where during definition of activity which starts subprocess, has to be provided definition version number). Process instance started in one version is completely realized in this version – there is no transferring process instance from one version to other if active version was changed – it could cause skipping important business steps).

2.4. Process Instance

Running process causes creation of instance of process definition. Instances are started in project context picked up by user. Project is entity representing real-world project in company. It is related with set of groups which members are working in this project. Now when instance is starting, decision in which group should first activity of process occur is taken. Simply the first group from process lanes groups, which match one of project groups is chosen. Further process execution depends of its interaction with user and certainly of process definition.

2.5. User Management

As it was mentioned applications of Workflow type, set in the company relations between business rules and organization structure (workers groups). So, to correct configuration of defined process model it is necessary to have tool for managing users account in system. In Tytan Workflow's administration application entities related with user management are:

- **user account** – objects representing given person, contain basic information about worker, such as personal data, email and also list of privileges profiles in particular groups.
- **project** – represents entity in which context process instances are started. User groups are related with projects.
- **group** – it has type and name, can be allocated to process lane and to project. There is possibility to create group hierarchy, therefore specifying for given group its superior group. Within the group user privileges profiles are admitted. Grouping workers in groups, and creating groups tree it is easy to reflect company structure with departments and smaller project teams.
- **privileges profiles** – they describe set of activities which can be performed by person with account to which given profile in given group is added (e.g. ability to create process instance, to browse process history, to allocate task to other workers). Opposite to groups profiles have flat structure and defines activities only in context of group.

2.6. User Interface

Tytan Workflow is available from all over the world through web browser. After log in user can interact with process instances which during its execution was allocated to him. Basic definition which should be explained here is task. Task is element of process instance – activity instance, allocated to concrete user, meaning that given action (e.g. *Provide order data*) should be performed by this worker. Main part of user interface is exactly task list, which shows tasks allocated to user, displayed as a list with basic informations like process instance id, subject, task name, due date (if it was set up) etc. Task may be edited, then are displayed all parameters attached to this activity during process definition, which can be changed by user (if its definition allows this – if they are not read only). After pursuing actions described by given task and after filling mandatory parameters (if they exists) user can initiate executing of next step of process, which most frequently results in showing other task in task list of another user (certainly excluding situations when given task was last in process and after its executing process ends). User has access to process instance history (if he has suitable privileges) – he can browse past steps of process, in which place his task is placed, who executed last tasks, when it was happened etc. He can also allocate task to other worker (set of potential candidates to take over the task is narrowed by groups attached to project and to lanes of process and also by user privileges). Besides user has access to various helpful tools like e.g. statistics of his tasks, process search (which makes possible checking in what state are processes that do not visualize on task list as user task – e.g. because they have ended), process creator to starting new processes instances, browser of user groups and few others.

2.7. Integration with 3rd Party Systems

Tytan Workflow system can be easily integrated with 3rd party systems using a set of open interfaces like Web Services, RMI, SOAP JMS and XML files, which make possible bi-directional communication. Currently Tytan is integrated with Requirement Management System RMST and with tool for report generation – Ocean.

3. Using Tytan Workflow in Software Engineering

Tytan Workflow application was used in ComArch S.A. to automate processes related directly with software development and maintenance. Examples of processes which was modeled and implemented are:

- software service
- change request management
- project management
- reports generation

3.1. Service Processes

It is set of few processes that are used in realization of notifications and problems handling (e.g. software bugs, suggestion, questions) that are reported by clients and

workers. Process maintain whole life cycle of notification called ticket – from its submitting through realization and finally until verification of solution or answer and closing ticket. There are for user groups that participate in this process and therefore there are four for each one (and also four group types):

- Clients – persons that create ticket and verify whether the solution is satisfying
- First line – it mediates between clients and rest of company workers, gathers information to describe problem more precisely, and if it can realize ticket
- Second line – occupy technical analysis, patches and corrections
- Production – if project is still in development changes that solves notified problem are attached into product or in its newest version

Each group has dedicated its own process – their instances are connected into hierarchy of subordinate and precedent processes, it means that e.g. first line process is started as subprocess of client process, and from the other hand process of second line is started as subprocess of first line process. Synchronization of data between processes is achieved with using mechanism of copying parameters values and also with help of global parameters which values are common for all processes in one hierarchy tree. Process participators from different groups may communicate between each other thanks to consultation mechanism, that provide possibility of sending text messages.

Most important steps of service processes are:

- **Client Process**
 - Registration – submitting person provide description of problem
 - Starting of first line subprocess
 - Waiting for ticket solution
 - Solution Verification – in case of rejecting the solution, process backs to first line
- **First line process**
 - Registration - accepting ticket parameters provided in client process
 - Problem solving
 - Starting of second line subprocess – it is not mandatory, takes place when solving the problem can not be achieved here
 - Solution verification – if negative process may back to second line or again to first line problem solving steps
 - Waiting for solution acceptance from client – if solution is rejected process backs to problem solving
- **Second line process**
 - Registration
 - Problem solving
 - Starting production process – not mandatory
 - Verification of solution
- **Production process**
 - Registration
 - Problem solving
 - Starting other production process instance – if somebody want to divide problem realization between wider group of people
 - Verification of solution

As you can see all process except of client's one are quite similar. But their strict separation is necessary – it provides possibility of division of work between specialized departments, gives also better control of whole process, makes easier creating statistics. Introducing such set of processes is step into *paperless* company. All data related to given ticket are provided through web browser as texts or attachments, and afterwards stored in database in records related to process instance, therefore improved management of data is possible. All advantages allows in some degree to reduce costs of whole process of software maintenance.

3.2. Change Request Management

This process is example of introducing ISO quality standards into company daily work. Process model was created basing on ISO procedure that concerns change request management. Change request is client's wish to change functionality of product which already has been implemented and works for some time. Changing functionality may be easy or very hard (that means expensive), has to be consulted with product road map to anticipate possible conflicts and many other factors influence whole process. It is necessary to provide proper handling of this kind of requests from client. There is several groups of workers that participate in this process:

- Attendance – people that contact with client, preparing and presenting offers an gathering opinions and answers
- Change manager – manage all change request from given product, allocates people resources which have to occupy given request
- Change request manager – is pilot of this project, inspects process of its realization
- Product manager – person which controls development of given product, circumscribe its future paths
- Analysts – prepare analysis of selected problems
- Workers – usually they got precise tasks what should be done

Main steps of change request management process are:

- Registration
- Initial analysis
- Consultation with client about costs
- Offer for client
- Realization
- Verification of changes

3.3. Project Management Process

This process model was implemented to provide better control of whole process of creating the project. Groups that participate in realization are:

- Registerer – person that enters informations about project and selects resource manager

- Resource manager – person that allocate people resources for project and selects project manager
- Project manager – controls whole procedure of project realization
- Worker – they execute tasks pointed by project manager

Project management consists of three processes:

- Contract Order Process – main process that serves as monitor during project realization
- Internal Project Order – this is subprocess started when realization of project requires ordering another project
- Task Order Process – this process is used by project manager to order particular tasks for workers

3.4. Report Process

This process is example of integrating Tytan Workflow with external systems – in this case with Ocean module for report generating. Process definition contains various activities that correspond with particular reports definition (which are stored in separate repository). User who opens such process instance can select one of defined reports type and in next task fill required parameters. Entered data are used as arguments in invoking procedures from Ocean to generate chosen report which is returned to user.

4. Summary

Using Tytan Workflow allows to increase in work efficiency through strict definition, organization and automation of company activities. Allows to speedup execution, centralize information storage, divide responsibility between company departments. It simplifies control of such aspects like work flow, times spent on particular stages of processes (thus could be helpful in its improvement) and efficiency of particular workers. It is also promising tool for integration various systems into common platform. Within one process definition many products may be used for particular actions. This all could be achieved under one very important condition – process model and its definition has to be simply good – fulfill goals for which it was projected. It is not simple matter – process definition is long and laborious operation – it require many consultation with workers to recognize and understand their work. But without good process model advantages from using Workflow software could easy turn into financial losses.

Bibliography

- [1] G. Alonso, D. Agrawal, A. ElAbbad and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. IEEE Expert 1997. Available on the Web at <http://www.almaden.ibm.com/cs/exotica/wfmsys.ps>
- [2] The Workflow Reference Model. Workflow Management Coalition, December 1994. Available on the web: <http://www.aiai.ed.ac.uk/WfMC/>

- [3] IEEE: IEEE Standard Glossary of Software Engineering Terminology, IEEE, Standard IEEE Std 610.12-1990, 1990
- [4] Solecki, A.: Workflow narzędziem z przyszłością. Strategie informatyzacji – Integracja systemów, InfoVide, 5 April 2000
- [5] Bennett, K.: Software Maintenance – Cut the biggest IT costs. Computer Bulletin, BCS, January 2005
- [6] Somerville, I.: Inżynieria oprogramowania (Software Engineering), *Wydawnictwo Naukowo Techniczne*, 2003
- [7] ComArch S.A.: Tytan Workflow – Increasing revenue by reducing complexity, 2004

Architecture of Parallel Spatial Data Warehouse: Balancing Algorithm and Resumption of Data Extraction

Marcin GORAWSKI

Silesian University of Technology,

Institute of Computer Science,

Akademicka 16, 44-100 Gliwice, Poland

e-mail: Marcin.Gorawski@polsl.pl

Abstract. In this paper we present a Parallel Spatial Data Warehouse (PSDW) system that we use for aggregation and analysis of huge amounts of spatial data. The data is generated by utilities meters communicating via radio. The PSDW system is based on a data model called the cascaded star model. In order to provide satisfactory interactivity for PSDW system, we used parallel computing supported by a special indexing structure called an aggregation tree. The balancing of a PSDW system workload is very essential to ensure the minimal response time of tasks submitted to process. We have implemented two data partitioning schemes which use Hilbert and Peano curves for space ordering. The presented balancing algorithm iteratively calculates optimal size of partitions, which are loaded into each node, by executing a series of aggregations on a test data set. We provide a collection of system tests results and its analysis that confirm the possibility of a balancing algorithm realization in proposed way. During ETL process (Extraction, Transformation and Loading) large amounts of data are transformed and loaded to PSDW. ETL processes are sometimes interrupted by occurrence of a failure. In such a case, one of the interrupted extraction resumption algorithms is usually used. In this paper we analyze the influence of the data balancing used in PSDW on the extraction and resumption processes efficiency.

Introduction

During the last 2 years deregulation of energy sectors in the EU has laid the foundations for a new energy market as well as created new customers for electrical energy, natural gas, home heating, and water. Those changes permit consumers to freely change the utilities deliverer. The most crucial issue in this regard is the automated metering of utilities customers' usage and the fast analysis of terabytes of relevant data gathered thusly. In case of electrical energy providers, meter reading, analysis, and decision making is highly time sensitive. For example, in order to take stock of energy consumption all meters should be read and the spatial data analyzed every thirty minutes. Relational analytical on-line processing (ROLAP) for spatial data is performed in data warehouses based on the cascaded star model [1]. Such data warehouses are called Spatial Data Warehouses (SDW) [9].

In the traditional approach, the ROLAP application contains hierarchies for individual dimensions. In order to reduce query answer time, the data is aggregated (materialized) on various levels of those hierarchies. In the case of spatial hierarchies

we do not know in advance all the hierarchies – in defining the query, a user can choose any piece of the region in the map. Traditional range queries executing first classifies spatial objects and then aggregates their attributes. Such an approach, however, is very time consuming, the reason being the goal is to obtain only a single value without classifying every object.

We propose an approach based on an aggregates index that supplements the traditional method with additional summary information in the intermediate positions. The most popular aggregates index are R-Tree [2], [21] and aR-Tree (aggregation R-Tree) [20], [23]. Therefore, in order to increase the efficiency of the SDW system we used the technique of materialized views in form of aggregation trees.

Another method to increase the SDW efficiency is data partitioning. Data partitioning requires the applying of the parallel system. The current parallel ROLAP approaches can be grouped into two categories: a) work partitioning [3], [4], [19] and b) data partitioning [2], [3], [5], [6], [17], [22].

In this paper, we study data partitioning methods for ROLAP cases of SDW and implement the data distribution in a well-scalable Parallel Spatial Data Warehouse (PSDW) architecture.

The main objective of a PSDW distribution is parallel execution of following operations: data loading, indices creation, query processing, resumption of ETL process. The data gathered in a PSDW is stored among many nodes, each one is maintained by an independent RDBMS.

There are a few methods of increasing parallel systems performance: tree-based data structures, grid files, chunking. The tree-based data structures are used for declustering of a data set making a data cube and to splitting it into a number of partitions. Partitions are in turn allocated to distributed system nodes. Grid files or cartesian product files are very strong and important techniques of improving distributed systems performance. A degree of parallel processing is determined by a data distribution amongst nodes, called Data Allocation Strategy [24].

Our previous solutions of the PSDW system balancing concerned on the case in which the system consisted of nodes having the same parameters or the same performance characteristics [10]. The essence of the problem considered in this work is PSDW system balancing (later called B-PSDW) which is based on computers with different performance characteristics [11]. The differences concern: computational capacity, size of operational memory and I/O subsystem throughput.

During ETL process (Extraction Transformation and Loading) large amounts of data are transformed and loaded to a data warehouse. It takes a very long time, several hours or even days. There is usually a relatively small time window fixed for a whole extraction. The more data to be processed, the longer the ETL process. When for some reason an extraction is interrupted, for example by hardware failure or no power supply, the extraction must be restarted. Such a situation is not rare. Sagent Technologies reports that every thirty extraction process is interrupted by a failure [25]. After an interruption there is usually no time left for running the extraction from the beginning. In this case, the most efficient solution is to apply one of the interrupted extraction resumption algorithms. In this paper we analyze the standard and our modified Design-Resume [15] algorithm (DR), and a combination of DR and staging technique (hybrid resumption). The modified DR handles extraction graphs containing many extractors and many inserters.

The ETL-DR environment succeeds the previous ETL/JB (JavaBeans ETL environment) and DR/JB (ETL environment with DR resumption support). The new

ETL-DR environment is a set of Java object classes, used to build extraction and resumption applications. This is analogous to JavaBeans components in the DR/JB environment. In DR/JB we implemented an estimation mechanism detecting cases where the use of DR resumption is inefficient. Unfortunately, the model we used did not take into account many significant external factors, like virtual memory usage. In the ETL-DR the situation changed. We improved the implementation of the DR filters, which resulted in reduction of resumption inefficiency. Now the resumption is almost always faster than restarting the extraction from the beginning. Hence, we decided to stop research on this mechanism. Another direction of our researches is combining the DR resumption with techniques like staging and checkpointing. The combination of DR and staging, we named it hybrid resumption. This approach performs better than pure DR algorithm.

In PSDW systems it is important to utilize the computing power optimally. The more differs the performance of machines the warehouse consists of, the more complicated the balancing is. We decided to use the partitioning algorithm for fact tables only. All the other dimension tables are replicated on the machines without any modification [8]. Such a balancing algorithm turned out to work satisfying according to our expectations. Now we want analyze its influence on the efficiency of the extraction process.

The remaining part of the paper is organized as follows: in the next section we briefly describe the general B-PSDW system architecture explaining the motivation for our work. Section 2 provides a description of our algorithm of balancing, and presents and discusses results of performed tests. Section 3 describes hybrid algorithms and ETL process test results. Finally Section 4 concludes the paper.

1. Motivation

Comprehensive and precise spatial telemetric data analysis can be performed by use of the technology called Automatic (Integrated) Meter Reading (A(I)MR) on the one hand, and data warehouse based Decision Support Systems (DSS) on the other [13]. Parallel Spatial Data Warehouse (PSDW) is a data warehouse gathering and processing huge amounts of telemetric information generated by the telemetric system of integrated meter readings. The readings of water, gas and energy meters are sent via radio through the collection nodes to the telemetric server (Figure 1). A single reading sent from a meter to the server contains a precise timestamp, a meter identifier, and the reading values. Periodically the extraction (ETL) system loads the data to the database of PSDW.

Our solution consists of two layers: the first is the Telemetric System of Integrated Meter Readings and the second is the Parallel Spatial Data Warehouse.

The PSDW system has a shared nothing type distributed architecture, where every node has its own processor, a memory area, and a disc. This architecture requires more sophisticated management, but offers almost linear system speedup [14]. Every node of the PSDW system is maintained by an independent Oracle 9i RDBMS.

All the communication between the nodes is realized only through network connections (message passing) with RMI technology (Remote Method Invocation).

The source data for the PSDW system are partitioned among N nodes, so that every node possesses the same data schema. A fact table, which usually takes more than 90% of the space occupied by all the tables, is split into N fragments with fixed

sizes. The dimension tables are replicated into all nodes in the same form [3]. The distributed processing phase is executed in each node only upon a data fragment stored in it.

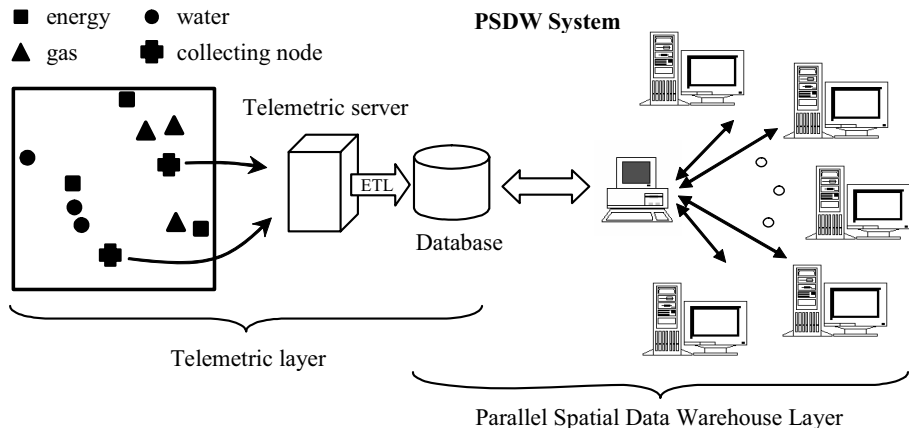


Figure 1. Cooperation of the data warehouse and telemetric system

To improve system's efficiency, an aR-tree with a virtual memory mechanism was implemented in the system. The index is constructed in a way that aggregates are computed as they arrive during query execution (unless they were computed in a response to previous queries). If the query is asking about data already aggregated they are received from the special system table. The index organization cause that the query determines the data that will be drawn to an aggregation. The next feature of this index structure is that the time taken to draw data already aggregated is very small in comparison to the time taken for aggregation, when large amounts of detailed data are pulled out from the data warehouse.

Our previous solutions for balancing the SDW system concerned a case in which the system consisted of nodes having the same parameters or the same performance characteristics. The essence of the problem considered in this work was the PSDW system balancing (later called B-PSDW) which is based on computers with different performance characteristics. The differences concern the computational capacity, size of operational memory, and I/O subsystem throughput.

2. The B-PSDW Balancing Fundamentals

The average response time of tasks submitted to process is minimized in balancing process in order to increase an overall efficiency of B-PSDW system. Average response time is defined as a average interval between a moment when job was submitted to a system and a moment when it leaves after processing.

In the B-STDW system each node performs the same query only upon a piece of data stored in it. Size of the piece must be properly selected to ensure the same work time of a particular system nodes. In the case of B-STDW systems based on computers

with the same parameters balancing is treated as ensuring the same load of every disk. This is achieved by using a proper data allocation scheme. The main task of the scheme is to guarantee that every node has the right contribution in evaluating different type of queries. All the methods presented above are designed to deal with the situation when a parallel system consists of identical nodes. In that case improving system parallelism means ensuring that every node has the same computational load. The B-STDW system is composed of nodes having different computational characteristics and this feature makes it impossible to apply mentioned methods directly.

In this paper we present a balancing method that is based upon setting the size of partitions that are loaded into system nodes (the fact table is partitioned). We propose a way of source dataset splitting and an algorithm of setting the partition size that enable us to achieve the small overall system's imbalance.

Choosing this approach to balancing we must deal with two inseparable problems: what is the best method of data allocation between the system nodes and how to set particular dataset sizes to ensure the same work time for each node. The next problem is how to link the sizes with node characteristics together.

To allocate data to nodes we exploited some well known data allocation schemes using space filling curves. The curve visits every point of space exactly once and never crosses itself, it could be used for the linear ordering of k-dimensional space [7]. Its next feature is good distance preserving mapping. In these schemes the curve is used to select next grid block which is in turn allocated to a node in round-robin fashion. The idea is based on the fact that two blocks that are close in linear space should also be close in k-dimensional space, thus they should be allocated on different nodes in order to ensure better parallelism of query execution [6], [18]. We are checking two types of curves: Peano and Hilbert [7], in addition we changed a schema fashion in a way which enable us to control the dataset size stored in each node. To mark nodes efficiency (and to link node characteristics with its dataset sizes) we propose a realization of single range query. The realization of such query must draw and aggregate all detailed data stored in system nodes. The measure of efficiency is the obtained aggregation time. The test set (used in balancing) is a subset of the fact table. The proposed algorithm iteratively calculates partition sizes executing a series of loading and aggregation (query execution). Due to algorithm's nature it can be used only before the main warehouse run, when system is tuned up. We have made some additional presumptions: the data warehouse is static (no data updates will be performed), characteristics of the nodes are constant and communication costs are very low (omitted).

2.1. Balancing Algorithm

Let us introduce the concept of a fact table division factor denoted by π_i [3]. Its value is a part of the fact table stored in the i node of the distributed warehouse system. Basing on the aggregation time, the π_i factor for each machine is iteratively computed. The goal of balancing is to obtain node work time similar to the mean work time. The balancing algorithm goes as follows:

1. Load dimension tables into all nodes in the same form.
2. Set all fact table division factors to $\pi_i = 1/N$, where N is the number of machines the warehouse consists of.
3. Load a test subset of fact table, partitioned according to the division factors into all nodes.

4. Perform test aggregation of the loaded set.
5. Compute imbalance factors. If maximum imbalance is smaller than assumed value go to 7, otherwise go to 6.
6. Correct division factors π_i using the aggregation time values, go to 3.
7. Load a whole fact table, partitioned according to the last computed division factors into all nodes.

In the first step dimension tables are loaded. The same data set is loaded into each node. The initial values of π_i factors are set to $1/N$ where N is the number of nodes. Next step of the algorithm is calculation of a fact table partition plan:

1. Calculate H/Z-value for the localization of each meter (see Section 4), where H denotes Hilbert curve and Z denotes Peano curve.
2. Sort meters according to H/Z-values in ascending order.
3. Allocate chunks to nodes using the round-robin method.

After loading the test data set, the test aggregation is performed and aggregation times are collected. In the next step the imbalance factors are computed in reference to the shortest time measured. If the maximum imbalance exceeds the assumed limit, the corrections are made to division factors and the process repeats. When the imbalance is small enough, then a final loading of a complete data set is run.

2.2. Tests

The system presented in this paper along with described balancing algorithm was implemented and installed on an infrastructure consisted of six PC computers connected via LAN network (five of computers were working as a server and one of them as a server/maintainer). The source data was stored on a node, which was playing the role of a system maintainer: realized the above-mentioned algorithm and all system access operations. Server nodes perform operations that were delegated by the maintainer. System configuration was as follows: 4 nodes - 2.8 GHz Intel Pentium 4, 512 MB RAM, 120GB 7200 RPM IDE HDD, 1 node - 1.7 GHz Intel Pentium 4, 256 MB RAM, 80GB 7200 RPM IDE HDD. The node working as a server and maintainer had following configuration: 3.2 GHz Intel Pentium 4, 1 GB RAM, 120GB 7200 RPM IDE HDD. All computers worked under Microsoft Windows XP, had an installed Java Virtual Machine v. 1.4.2_04 and an Oracle 9i server. System was connected together by 100Mbit Ethernet network.

The fact table stored in system had about 9.4 million rows and contained measures data of 850 media utility meters from one year period. The test subset of the table was used in balancing processes had 5.6M rows, i.e. measures of 510 meters. Twelve balancing processes were performed for the described system. Each of processes used different combination of correction factors and ordering variants and was followed by a three different range queries (about aggregates from some spatial region and time window). The first query asked about the aggregates from a few large regions, the second about a dozen or so small regions and the third drawn aggregates from a few large regions and a few small regions. All regions in those queries were randomly located on the map. Each one query was performed in a three variants: for 31, 59 and 90 days aggregation period.

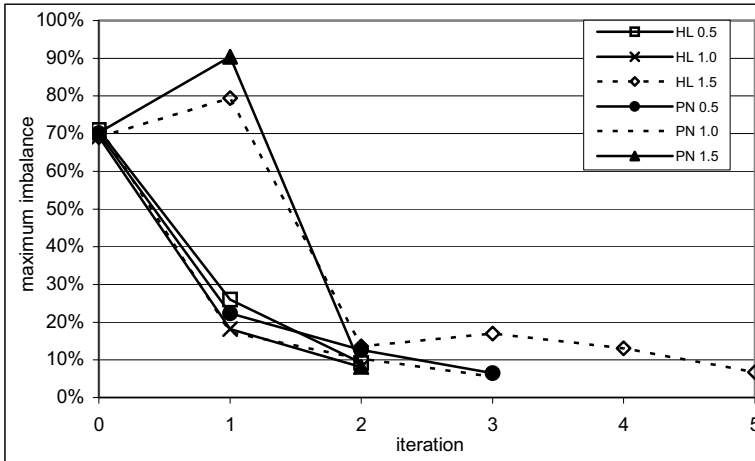


Figure 2. Charts of maximum imbalance for balancing processes with using test subset

We begin the result analysis with comparison of maximum imbalance graphs (maximum imbalance amongst all of node imbalances per process iteration) for test subsets (Figure 2). The process stopped when imbalance went below 10%. Performed tests checked the influence of used: test subset size, variant of space filling curve and values of correction factors on duration of the balancing process. Obtained results shows that the balancing processes with using of the Hilbert curve in generality were longer in duration with respect to corresponding processes when the Peano curve was used. Another observation is the fact that higher values of the correction factors result in shortening of balancing process.

3. Resumption Algorithm

Resumption algorithm (Redo group) enriched by the possibility of handling ETL processes containing many loading nodes. The key feature of this algorithm is that it does not impose additional overhead on normal ETL process. The Design-Resume [15], [16] algorithm (DR) works using properties assigned to each node of the extraction graph and data already loaded to a destination prior to a failure. This algorithm belongs to the group of redo algorithms, but during resumption it uses additional filters that remove from the data stream all tuples contributing to the tuples already loaded. An extraction graph is a directed acyclic graph (DAG), whose nodes process data and whose edges define data flow directions.

The algorithm is divided into two phases. In phase 1 the extraction graph is analyzed and additional filters are assigned (Design procedure). In phase 2 the filters are initialized with data already loaded into a data warehouse (Resume procedure). The most important feature of the DR algorithm is that it does not impose any additional overhead on the uninterrupted extraction process. Algorithms like staging or save-pointing usually increase processing time a lot (even several times). The drawback of the DR algorithm is that it cannot be applied if before a failure no output data were produced. Design procedure is usually run once to assign filters that are needed during

resumption. The Resume procedure is run each time the ETL process must be resumed. It fetches data already loaded from a destination, and uses the data to initialize additional filters assigned in phase 1. The basic DR algorithm can resume only single-inserter extraction graph. This limitation lowers the resumption efficiency, and encouraged us to modify the algorithm [21].

3.1. Hybrid Resumption

General idea of the hybrid resumption is a combined resumption algorithm taking advantage of both the Design-Resume algorithm and the staging technique [12]. Similar approach combining the DR and savepoints was examined in [15]. During a normal extraction process, each node selected by the graph designer generates output data and sends it to the consecutive nodes, and also writes each output tuple to a temporary disk file. When the node finishes processing, the file it generates is marked as containing a complete data set.

Extraction may be interrupted by a failure before or after the entire transitional data file (or files) is written. When failure occurs without completing the file, only a standard DR resumption can be applied. But, if the file is marked as containing a complete data set, hybrid resumption can be used. In the first step, from the extraction graph are removed all the nodes that correspond only to the nodes working in staging mode. These nodes are unnecessary because they do not need to be run again to get already-written transitional data. Next, the node that generated the file is replaced with a source reading the transitional data file. This way the saved data can be processed without repeating its generation process. This of course can save a lot of time. Just behind the source reading the transitional data DR filters can be used. This way we combine two algorithms: staging and DR. The staging part reduces the number of nodes that has to be run during resumption, and the DR part reduces the amount of data that must be processed.

3.2. Tests

The base for our tests is an extraction graph containing 4 extractors and 15 inserters (loading nodes). The graph consists of three independent parts, but it is seen by the extraction application as a single ETL process. To load data into a PSDW consisted of 5 PC machines we had to increase the number of inserters in the graph. Each inserter loads data into a single database table. Finally we obtained the graph containing 75 inserters.

The ETL process generates a complete data warehouse structure (Figure 3). It is a distributed spatial data warehouse system designed for storing and analyzing a wide range of spatial data [10]. The data is generated by media meters working in a radio based measurement system. All the data is gathered in a telemetric server, from which it can be fetched to fill the data warehouse. The distributed system is based on a new model called the cascaded star model. The test input data set size is 500MB.

The tests were divided into two parts. First the balancing process were run and the parameters for distribution filters were obtained. Then we run a complete extraction process, loading both dimension and fact tables. The fact tables were distributed among the machines according to the balancing results.

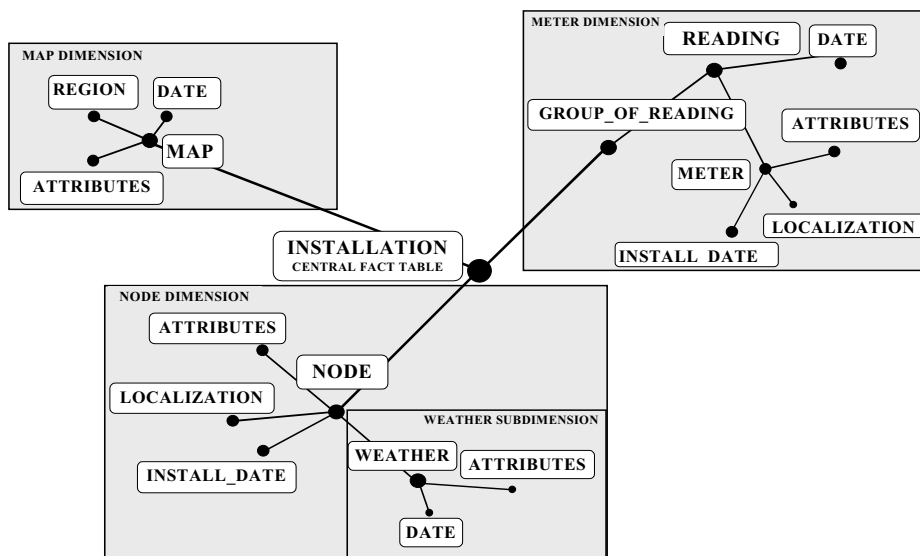


Figure 3. Schema of the generated data warehouse

During each loading test the extraction process was interrupted in order to simulate failure. The resumption process was then run and the time was measured. The collected measurement results permitted us to prepare resumption charts showing the resumption efficiency depending on the time of a failure.

The goal of the tests is to examine influence of the balancing algorithm applied to our distributed warehouse system on the extraction process. We tested pure Design-Resume resumption algorithm and a hybrid resumption which is a combination of DR and staging technique. May conclude that the use of balancing has a marginally low influence on the performance of the extraction process (Figure 4). The advantage of the balanced extraction does not exceed a few percents. Similarly there is no visible influence on the efficiency of the resumption process. Only the difference between pure DR and hybrid resumption is a slightly bigger.

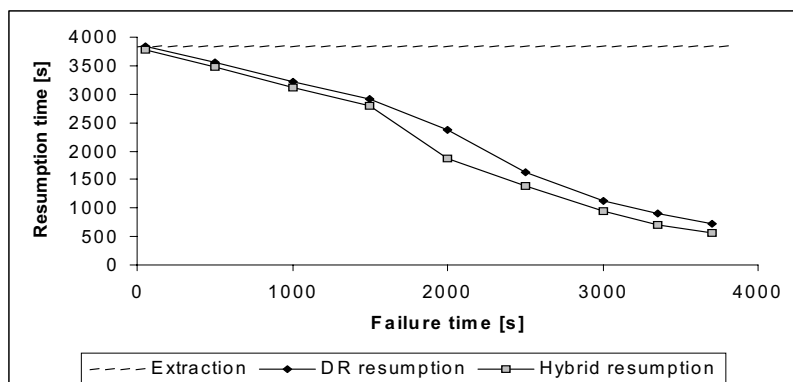


Figure 4. Efficiency of resumption when fact table distribution relies on the balancing results

The reason for such results is in our opinion the single computer extraction process. No matter if we balance the system or not. The bottleneck of the system during the extraction is the PC running the extraction process. The other computers are fast enough to work efficiently even with unbalanced data. Probably to take advantage of the balancing, the extraction must become a distributed process. This should avoid a single PC being a bottleneck of a whole system.

4. Summary

In this paper we presented iterative algorithm of balancing parallel data warehouse system load designed for systems having heterogeneous nodes characteristics. Any input information about system architecture nor user interaction during balancing process is not required by the algorithm. To calculate nodes partitions, the algorithm takes only subset of dataset stored in the system.

Provided test set of the algorithm implementation in real PDSW system confirms the possibility of a balancing realization in proposed way. The comparison of results for Hilbert and Peano curves shows that the second is more suitable for balancing purposes in the described system. The results show that balancing processes with using this curves in general were shorter in comparison with corresponding processes using Hilbert curve. Moreover, Peano curve dealt better with higher values of correction factors.

The increasing of test subset size results in decreasing of balancing process iteration number. Moreover, along with growing of test subset size the imbalance of queries submitted to process lowers. In general small region queries have greater imbalances than large one. The reason is the fact that for small regions there is higher possibility of unequal distribution of data amongst nodes (in relation to nodes efficiency).

The Design-Resume algorithm designed by Labio et al was briefly explained. Its general idea was presented and a few drawbacks were pointed out. We focused on the problem of resumption, when the extraction graph contains many loading nodes, and modified the algorithm to handle such cases more efficiently. We examined a performance of the ETL process depending on the data distribution method. There were two possibilities: data distributed uniformly among system nodes according to the meter number, or balanced data distribution, where the balancing goal was to obtain the fastest response time of the system based on the warehouse.

As shown in the tests, the influence of the balancing on the ETL process is very low. We observed reduction of the extraction process time by 1-2%. The reason for this is in our opinion the single machine extraction process. In our tests the ETL software runs on one PC and the 5 other PCs are warehouse system nodes running an Oracle database. We suspect that distributing the extraction process on the larger number of machines could bring more advantages. This is going to be the next step of our research.

References

- [1] Adam, N., Atluri, V., Yesha, Y., Yu, S. Efficient Storage and Management of Environmental Information, IEEE Symposium on Mass Storage Systems, 2002.
- [2] Beckmann, B., Kriegel, H., Schneider, R., Seeger, B. The R*-tree: An Efficient and Robust Access Method. SIGMOD, 1990.

- [3] Bernardino, J., Madera, H. Data Warehousing and OLAP: Improving Query Performance Using Distributed Computing. CAiSE*00 Conference on Advanced Information Systems Engineering, Sweden, 2000.
- [4] Chen, Y., Dehne, F., Eavis, T., Rau-Chaplin, A. Parallel ROLAP Data Cube Construction On Shared-Nothing Multiprocessor .Proc. 2003 International Parallel and Distributed Processing Symposium (IPDPS2003), France, 2003.
- [5] Datta, A., VanderMeer, D., Ramamritham, K. Parallel Star Join + DataIndexes: Efficient Query Processing in Data Warehouses and OLAP. IEEE Trans. Knowl. Data Eng. 2002.
- [6] Dehne, F., Eavis, T., Rau-Chaplin, A. Parallel Multi-Dimensional ROLAP Indexing. Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), Japan, 2003.
- [7] Faloutsos C. and Bhagwat P.: Declustering using fractals in Proc. of the Int'l Conf. on Parallel and Distributed Information Systems, San Diego, California, January 1993.
- [8] Gorawski, M., Chechelski, R., Spatial Telemetric Data Warehouse Balancing Algorithm in Oracle9i/Java Environment, Intelligent Information Systems, Gdansk, Poland, 2005.
- [9] Gorawski, M., Malczok, R. Distributed Spatial Data Warehouse. 5th Int. Conf. on Parallel Processing and Applied Mathematics, PPAM 2003, Czestochowa, *Springer-Verlag*, LNCS3019.
- [10] Gorawski, M., Malczok, R., Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation Tree. 5th Workshop on Spatial-Temporal DataBase Management (STDBM_VLDB'04), Toronto, Canada 2004.
- [11] Gorawski, M., Malczok, R.: Materialized aR-Tree in Distributed Spatial Data Warehouse, 15th ECML/PKDD_SSDA: Mining Complex Data Structures, Pisa, Italy 2004.
- [12] Gorawski, M., Marks, P.: High Efficiency of Hybrid Resumption in Distributed Data Warehouses, 1th Workshop: High Availability of Distributed Systems (HADIS_DEXA05) Copenhagen, Denmark August 2005.
- [13] Han, J., Stefanovic, N., Koperski, K. Selective Materialization: An Efficient Method for Spatial Data Cube Construction. In Research and Development in Knowledge Discovery and Data Mining, Second Pacific-Asia Conference, PAKDD'98, 1998.
- [14] Hua K., Lo Y., Young H.: GeMDA: A Multidimensional Data Partitioning Technique for Multiprocessor Database Systems. Distributed and Parallel Databases, 9, 211-236, University of Florida, 2001.
- [15] Labio W., Wiener J., Garcia-Molina H., Gorelik V., Efficient resumption of interrupted warehouse loads, SIGMOD Conference, 2000.
- [16] Labio W. J., Wiener J. L., Garcia-Molina H., and Gorelik V. Resumption algorithms. Technical report, Stanford University, 1998. Available at <http://www-db.stanford.edu/wilburt/resume.ps>.
- [17] Martens, H., Rahm, E., Stohr, T. Dynamic Query Scheduling in Parallel Data Warehouses Euro-Par, Germany 2002.
- [18] Moore D. Fast Hilbert curve generation, sorting, and range queries
- [19] Muto, S., Kitsuregawa, M. A Dynamic Load Balancing Strategy for Parallel Datacube Computation. DOLAP 1999.
- [20] Papadias, D., Kalnis, P., Zhang, J., Tao, Y. Efficient OLAP Operations in Spatial Data Warehouses. Proceedings of the 7th International Symposium on Spatial and Temporal Databases, (SSTD), Springer Verlag, LNCS 2001.
- [21] Rao, F., Zhang, L., Yu, X., Li, Y., Chen, Y., Spatial Hierarchy and OLAP – Favored Search in Spatial Data Warehouse. DOLAP, Louisiana, 2003.
- [22] Rohm, U., Bohm, K., Schek, H.J., Schuldt, H. FAS – a Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Component. Proceedings of the Conference on Very Large Databases (VLDB), Hong Kong, 2002.
- [23] Tao, Y., Papadias, D., Range Aggregate Processing in Spatial Databases. IEEE Transactions on Knowledge and Data Engineering (TKDE, 2004).
- [24] Yu-Lung Lo, Kien A. Hua, Honesty C. Young, GeMDA: A Multidimensional Data Partitioning Technique for Multiprocessor Database Systems, Distributed and Parallel Databases, Volume 9, Issue 3, May 2001.
- [25] Sagent Technologies Inc. Personal correspondence with customers

This page intentionally left blank

2. UML-based Software Modeling

This page intentionally left blank

Data Modeling with UML 2.0

Bogumiła HNATKOWSKA, Zbigniew HUZAR and Lech TUZINKIEWICZ

*Wrocław University of Technology,
Institute of Applied Informatics,
Wrocław, Poland*

e-mails: {Bogumila.Hnatkowska, Zbigniew.Huzar, Lech.Tuzinkiewicz}@pwr.wroc.pl

Abstract. Database design goes through three-stage development process: conceptual modeling, logical modeling and physical modeling. The paper deals with transformation of conceptual models to logical ones. Assuming that conceptual models are expressed in UML 2.0, the paper presents two groups of transformation rules. The first group contains commonly used rules dealing with transformation of conceptual models. The second group gathers specific rules, proposed by authors, that can be applied for new features of the UML 2.0.

Introduction

The UML is a modeling language used in object-oriented approaches to software development. Database modeling is a typical issue found in the software development processes. The UML offers mechanisms that may be used in database modeling. The mechanisms were strongly influenced by classic Entity-Relationship model [5], and its extensions [8], however, UML mechanisms provide more expressive modeling power [3], [4], [12], [13].

Traditionally, database design goes through three-stage development process: conceptual modeling, logical modeling and physical modeling [1], [5], [8], [13]. Conceptual and logical models that are results of the first two stages of database modeling are the subject of our interest. A conceptual model describing an application domain is a specification for a logical model. Logical model is assumed to be the relational one.

Class diagrams are usually applied in the conceptual modeling stage. The basic elements of database modeling are classes connected by various relationships. Associations and generalizations are the most important among these relationships. The logical model may be specified by a stereotyped class diagram [16] or by SQL language [2], [6].

The aim of the paper is twofold: the first one is to present new possibilities of UML 2.0 for conceptual modeling, and the second one is to propose rules transforming conceptual models into logical ones.

The paper is organized as follows. Section 1 reviews general principles of conceptual and logical modeling. Section 2 briefly presents new elements in the UML 2.0 facilitating conceptual modeling. Section 3 gives a systematic presentation of rules that can be used in the transition from conceptual to logical models. The last Section 4 summarizes the paper with concluding remarks.

1. Database Modeling

Database modeling goes through different abstraction levels, i.e. conceptual, logical, and physical [1], [5], [8], [13].

A conceptual model represents a modeled domain, and it is expressed by: a system glossary and class diagrams with some additional constraints that express static, and dynamic integrity demands [3], [5]. These constraints could be written in OCL or informally, in natural language, and usually are manifested by such artifacts as business rules [7].

A logical model also represents the modeled domain, but in terms of a chosen data model. In the paper the relational database model is assumed. The relational model is expressed by relations that are implemented by tables in database systems [14], [10]. These models may be specified by stereotyped class diagrams [16] or textually by SQL expressions [2], [6].

It is desirable for a conceptual model to consist of elements representing elementary (atomic) entities from the modeled domain. At this stage of database modeling it is recommended to decompose (normalize) complex entities [13] in order to achieve a logical model in the 3rd NF. An example of a conceptual model resulting from the process of class normalization is given in Figure 1. A document *invoice* could be seen at different detailed levels. On the left side of the figure the *Invoice* class is a complex entity. Its normalization — related to the *Invoice* class with dependency relationship — provides the package *Invoice* with four associated elementary classes, presented on the right side of the figure.

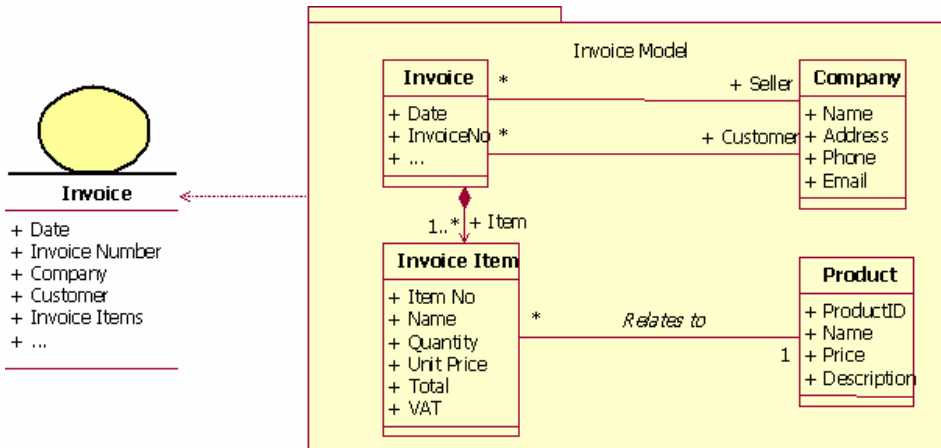


Figure 1. Example of class normalization

A logical model is derived from conceptual one by applying some mapping rules. Before the transformation process, a conceptual model should be defined at appropriate level of detail. It is assumed that for each class in the conceptual model:

- class name is an entry in system glossary,
- class attributes have specified names and types,

- domain constraints are defined,
- candidate keys are identified.

Additionally, all relationships among classes are defined.

2. New Features of Class Diagram in UML 2.0

In this Section, we present the main changes in class diagrams resulting from the new definition of association and new mechanisms related to generalizations.

2.1. Association

An association is a specification of a semantic relationship among two or more classes. An association in UML 2.0 is defined slightly different as in UML 1.x. Firstly, an association in UML 2.0 is a classifier. An instance of an association is called a link. A link is a tuple with one value for each end of the association. The value for each association end contains a reference to an object of the associated class. It contrasts to UML 1.x, where the link is a tuple of objects of associated classes. Secondly, semantics of an association in UML 2.0 is considered as a description of a collection of links. The collection of links may be a set, a bag or a list. Such a collection is called an extend of the association.

An association has an optional name, but most of its description is found in a list of its association ends, each of which describes the participation of objects of connected class in the association. Each association end specifies properties that apply to the participation of the corresponding objects, such as multiplicity, and other properties like navigability or aggregation that apply to binary associations only, but most properties apply to both binary and n -ary associations ($n > 2$).

The meaning of multiplicity is described as follows. For n -ary association, choose any $n - 1$ ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection [15]. If the association end is declared as unique, this collection is a set; otherwise this collection is a bag. If the association end is declared as ordered, this collection will be ordered.

The information about uniqueness and ordering of an association end will be represented by strings enclosed in curly braces near this end. The decoration {ordered} shows that the end represents an ordered collection, and {bag} shows that the end represents a bag, i.e. a collection that permits the same element to appear more than once. Figure 2.a illustrates an exemplary class diagram, where association *AssOL* describes a collection of links being a bag, and Figure 2.b presents an object diagram — an instance of the class diagram.

N -ary associations are further not considered.

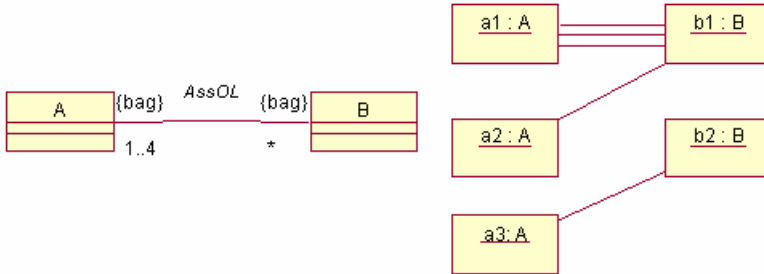


Figure 2. a) The class diagram, b) The object diagram

2.2. Generalization Set

Generalization in UML 2.0 is defined in similar way as in UML 1.x, i.e. as a taxonomic relationship between a more general and more specific element. The only difference is that generalizations may be defined between associations as the associations in UML 2.0 belong to classifiers.

A more general class (superclass) may be specialized into more specific classes (subclasses) along a given dimension, representing selected aspects of its structure or semantics. For example, taking sex into account a superclass *Person* may be specialized into two subclasses *Male* and *Female* representing persons of different sex (Figure 3). A set of generalizations representing such a dimension is called a generalization set. UML 2.0 generalization sets replace UML 1.x discriminators with roughly equivalent capability. The generalization set may have properties that are expressed in terms of the following standard constraints:

- `{disjoint}` meaning that any object of a superclass may be an instance of at most one subclass;
- `{overlapping}` meaning that any object of a superclass may be an instance of more then two subclasses;
- `{complete}` meaning that every object of a superclass must be an instance of at least one subclass;
- `{incomplete}` meaning that an object of a superclass may fail to be an instance of any subclass.

The constrains may be used individually or may be arranged in pairs: `{disjoint, complete}`, `{disjoint, incomplete}`, `{overlapping, complete}`, and `{overlapping, incomplete}`. The default constraint is `{disjoint, incomplete}`.

A given superclass may posses many generalization sets representing multiple dimensions of specialization. Each generalization set has its name shown as a text label following a colon on generalization arrow.

The example in Figure 3 exhibits two generalization sets *Sex*, and *Status*.

2.3. Power Types

Powertype is a new mechanism introduced by UML 2.0. It is a metaclass related to a given generalization set. Instances of the metaclass are subclasses of the same

superclass associated to a given power type class. For example, instances of the generalization set *Status* in Figure 3 are *Student*, *Teacher*, and *Employee*. The figure also exhibits the metaclass *Status* (labeled with «powertype») related to the generalization set with the same name. Multiplicity of the association end at *Status* metaclass is a consequence of constraints {incomplete, disjoint} imposed on the *Status* generalization set. It says that each object of the *Person* class may belong to at most one subclass.

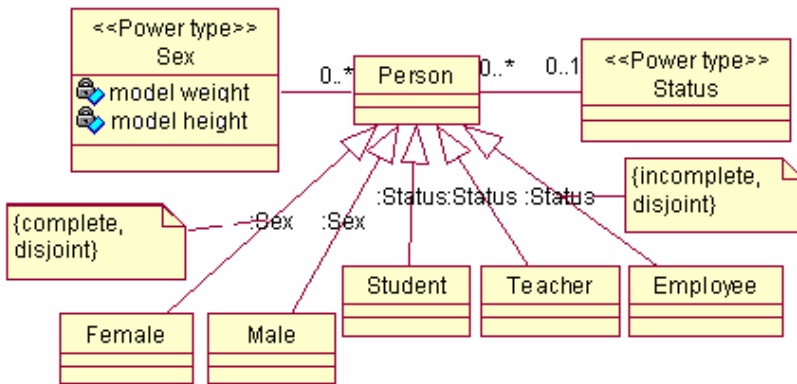


Figure 3. Generalization set and powertype example

The powertype adds the ability to declare properties that apply to individual subclasses in the generalization set. It means that the property assigned to a given subclass is a common property for all instances of the subclass. For example, the powertype *Sex* introduces two attributes: *model weight*, and *model height*. These attributes have some values, which are the same for all instances of the subclass *Female*, and other values, which are the same for all instances of the subclass *Male*.

3. Transformation Rules

This section presents how to transform a conceptual model into a logical model in a consistent way. There are two categories of transformation rules: general rules [3], [5], [8], [9], and specific rules. General rules take into account classes, two relationships (i.e. associations, and generalizations), and properties of these relationships, while specific rules take into account also properties of related classes.

The general rules of transformation of conceptual model into a logical model are as follows:

- G.1 A class from a conceptual model is transformed into a table.
- G.2 A class attribute is transformed into a column in the corresponding table provided that there exists a mapping from attribute type into a column type of the table.
- G.3 The table representing a class is assigned a primary key selected from a set of candidate keys. The candidate keys not chosen as primary ones are considered as alternative keys, and must fulfill the table constraint UNIQUE.

- G.4 A binary association of the many-to-many multiplicity type is transformed into a table. The primary key of the table consists of two foreign keys which are primary keys of tables representing associated classes.
- G.5 A generalization is represented by a foreign key of the table representing subclass. This foreign key is also a primary key. Its value indicates uniquely a row in the table representing the superclass.

We consider the transformation rules for binary association of the many-to-many multiplicity type. Figure 4 illustrates an application of rules G.1–G.4.

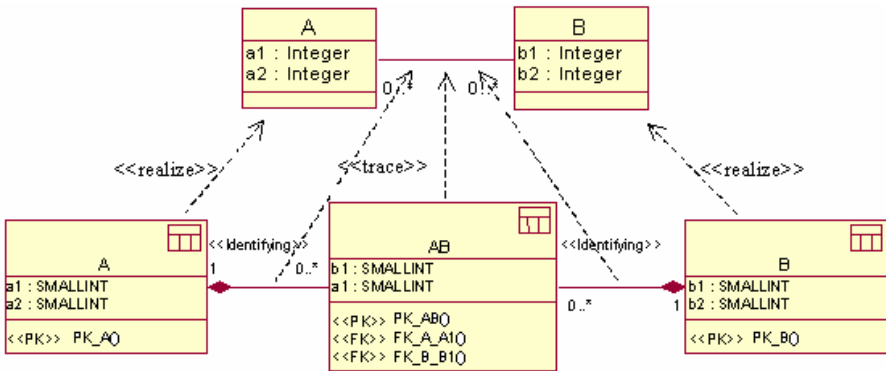


Figure 4. Transformation rules for binary, many-to-many association

In the example, we assume that the attributes a1 and b1 are candidate keys of classes A and B, respectively. The association is transformed into the table AB. The table is related to tables A and B by foreign keys. The result of the transformation — the logical model — is presented in the Rational Rose profile notation [16]. Note, that the notation used by Rational Profile for relationships between tables is not consistent with UML. The meaning of the filled diamond in the Profile is not consistent with composition relationship.

Alternative presentation form of the logical model in SQL notation is shown below:

```
CREATE TABLE AB (
a1 SMALLINT NOT NULL,
b1 SMALLINT NOT NULL,
CONSTRAINT PK_AB PRIMARY KEY (a1, b1)
);
CREATE TABLE B (
b1 SMALLINT NOT NULL,
b2 SMALLINT NOT NULL,
CONSTRAINT PK_B PRIMARY KEY (b1)
);
CREATE TABLE A (
a1 SMALLINT NOT NULL,
a2 SMALLINT NOT NULL,
CONSTRAINT PK_A PRIMARY KEY (a1)
);
ALTER TABLE AB ADD CONSTRAINT FK_A_A1 FOREIGN KEY (a1) REFERENCES A (a1);
ALTER TABLE AB ADD CONSTRAINT FK_B_B1 FOREIGN KEY (b1) REFERENCES B (b1);
```


The second example illustrates the application of the rule G.5 related to the transformation of generalization — see Figure 5.

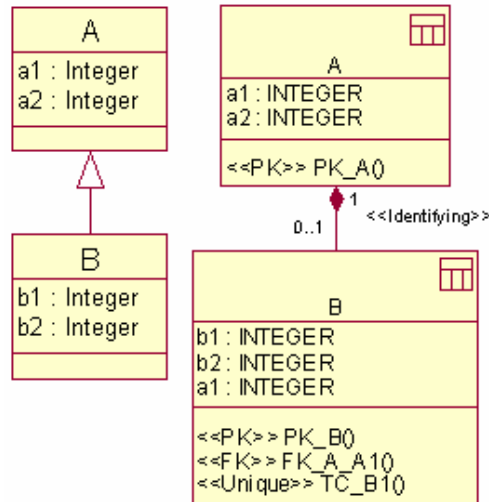


Figure 5. Application of G.5 rule for model transformation

```
CREATE TABLE A (
a1 INTEGER NOT NULL,
a2 INTEGER NOT NULL,
CONSTRAINT PK_A PRIMARY KEY (a1)
);
```

```
CREATE TABLE B (
a1 INTEGER NOT NULL,
b1 INTEGER NOT NULL,
b2 INTEGER NOT NULL,
CONSTRAINT PK_B PRIMARY KEY (a1)
);
```

```
ALTER TABLE B ADD CONSTRAINT FK_A_A1 FOREIGN KEY (a1) REFERENCES A (a1);
ALTER TABLE B ADD CONSTRAINT TC_B1 UNIQUE (b1);
```

Now, let us consider a new rule resulting from the {bag} as a new property of an association end. The respective rule is a modification of G.4 rule:

- S.1 A binary association of the many-to-many multiplicity type is transformed into a table. The primary key of the table is an artificial key. The primary keys of the tables representing associated classes are foreign keys in the table representing the association.

Figure 6 illustrates the application of the S.1 rule.

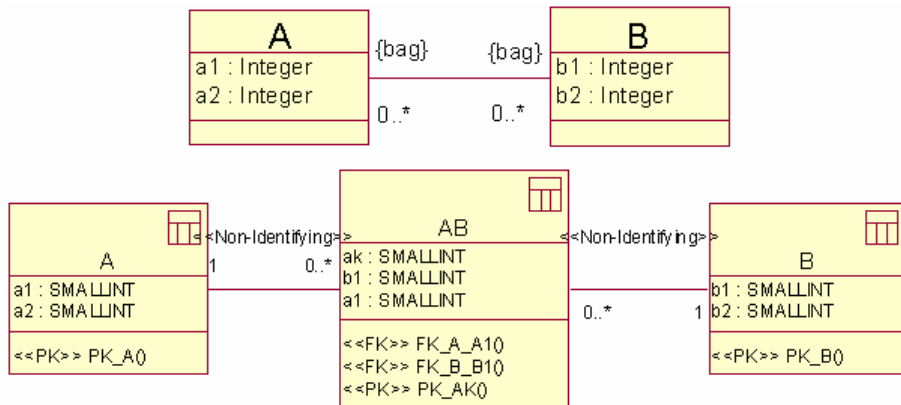


Figure 6. Multiset specification and their representation at logical level

The presence of {bag} at the association ends allows for duplicates of links between the same pair of objects. The links are not discriminated by primary keys of tables representing associated classes. Therefore, an artificial key is necessary to represent these links. The logical model in textual form is presented in the SQL listing below:

```
CREATE TABLE AB (
b1 SMALLINT NOT NULL,
a1 SMALLINT NOT NULL,
ak SMALLINT NOT NULL,
CONSTRAINT PK_AK PRIMARY KEY (ak)
);
```

```
ALTER TABLE AB ADD CONSTRAINT FK_A1 FOREIGN KEY (b1) REFERENCES B (b1);
ALTER TABLE AB ADD CONSTRAINT FK_A2 FOREIGN KEY (a1) REFERENCES A (a1);
```

When the multiplicity of at least one association end is $m..n$, where $m > 0$, and n is limited, and $n > 1$, then an additional check of consistency should be performed, for example, by a trigger.

Next specific rules are concerned with generalization. The following assumptions are anticipated. We do not consider generalizations with multi-inheritance. Transformation of a generalization set depends on the generalization set properties. The interesting ones are: (a) {complete}, (b) {incomplete, overlapping}, (c) {incomplete, disjoint}, (d) {complete, overlapping}, (e) {complete, disjoint}.

- S.2 A superclass with only one subclass and the property {complete} is represented by one table. Columns of the table represent attributes of the super- and subclass.
- S.3 A superclass with many subclasses and the property {overlapping} is realized according to the G.5 rule separately for each subclass. All tables representing subclasses are expected to have the same primary key. (In general, primary keys in tables representing different subclasses may have the same values.)

- S.4 A superclass with many subclasses and the property {incomplete, disjoint} is transformed according to the S.3 rule with the additional constraint that the set of values of primary keys of tables representing different subclasses must be disjoint.
- S.5 Transformation of a superclass with many subclasses and the property {complete, disjoint} depends on that if the super class is associated with other classes.
- if the superclass is not associated with other classes then the S.2 rule is applied, with the additional constraint that the set of values of primary keys of tables representing different subclasses must be disjoint;
 - otherwise, the S.4 rule is applied, with the additional constraint that each row in the table representing the superclass is referenced exactly by one foreign key of one of the tables representing subclass.

Transformation rules for generalization are explained by a simple example of the conceptual model, presented in Figure 7.

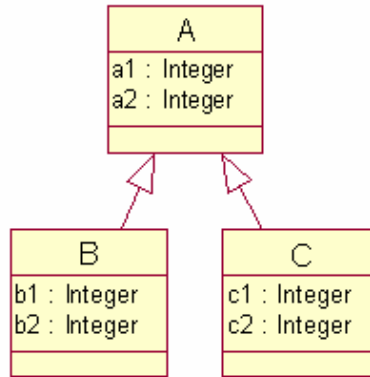


Figure 7. The conceptual model with generalization relationship between many subclasses

Figure 8 shows an example of a logical model resulting from applying the S.5a rule to the conceptual model presented in Figure 7.

The logical model is listed in SQL below:

```

CREATE TABLE B (
a1 SMALLINT NOT NULL,
a2 SMALLINT NOT NULL,
b1 SMALLINT NOT NULL,
b2 SMALLINT NOT NULL,
CONSTRAINT PK_B PRIMARY KEY (a1),
CONSTRAINT TC_B UNIQUE (b1)
);
CREATE TABLE C (
a1 SMALLINT NOT NULL,
a2 SMALLINT NOT NULL,
c1 INTEGER NOT NULL,
c2 INTEGER NOT NULL,
CONSTRAINT PK_C PRIMARY KEY (a1)
CONSTRAINT TC_C UNIQUE (c1),
);
  
```

The last specific rule relates to powertypes. We assume that names of generalization sets in a given conceptual model are unique, and at least one powertype is defined.

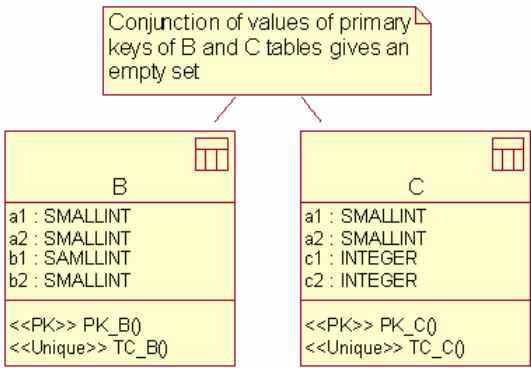


Figure 8. The logical model resulting from applying the S.5a rule to the conceptual model presented in Figure 7.

S.6 Powertypes in a conceptual model are represented by two tables in a logical model – see Figure 9.

- (a) The first table, called *PowerTypeFamily*, declares existence of powertypes and the respective superclasses. It contains two columns: *powertype name* (the primary key), and the *name* of related superclass.
- (b) The second table, called *PowerTypeInstances*, defines a set of instances of each powertype. It contains two columns: *powertype name*, and the *names* of subclasses – instances of the powertype. The primary key of the table is composed of these two names.
- (c) Optionally, when a given powertype has own properties, a new table for a such powertype should be defined. It is suggested that the name of the table be the name of the powertype, and the schema of the table reflects properties of this powertype. The primary key of this table is the attribute, whose values are all names of the powertype’s instances.

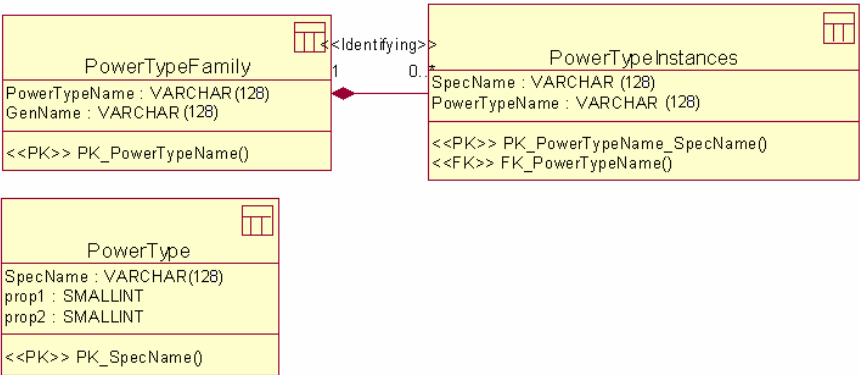


Figure 9. Power type representation in a logical model.

The logical model in textual form is presented in SQL listing below:

```
CREATE TABLE PowerTypeFamily (
PowerTypeName VARCHAR(128) PRIMARY KEY,
GenName VARCHAR(128) NOT NULL,
);
CREATE TABLE PowerTypeInstances (
SpecName VARCHAR(128),
PowerTypeName VARCHAR(128) REFERENCES PowerTypeFamily,
CONSTRAINT PK_PowerTypeName_SpecName PRIMARY KEY (PowerTypeName,
SpecName)
);
CREATE TABLE PowerType (
SpecName VARCHAR(128) PRIMARY KEY,
prop1 ...,
prop2 ...,
);
```

4. Conclusions

The paper presents the new features of UML 2.0 that seem to be interesting in data modeling. These features allow conceptual models to be more precise, and adequate to an application domain.

In database modeling, conceptual models are transformed into logical ones. The transformation is based on the set of commonly known rules [3], [5], [8], [9]. Extension of the UML entails the need of revision of existing transformation rules and elaboration of some new ones.

The paper presents two groups of transformation rules. The first group contains general, commonly used rules dealing with transformation of conceptual models written in UML 1.x [3], [5], [8]. The second group gathers specific rules that can be applied to new features of the UML 2.0.

We have limited our considerations to binary associations, and to generalizations with single inheritance which are the most frequently used in practice. We have not taken into account abstract classes, but transformation rules for generalization with {complete} property can be adopted for them.

In the paper we used two different notations for logical model presentation: Rational Rose Database profile, and SQL99 standard.

The application of the proposed transformation rules to many examples has convinced us that they preserve information contained in conceptual model.

The proposed transformation rules can be easily implemented in existing CASE tools.

The UML 2.0 specification document [15] has not been formally approved. While analyzing it, along with publications presenting UML 2.0, e.g. [11] some interpretation problems of UML 2.0 features have been encountered. For example, the problem is how to interpret multiplicity specified at the end of association labelled by bag property. It is not clearly defined whether the multiplicity is interpreted either as the limit of associated objects or as the limit of objects' references. Depending on the chosen interpretation two different logical models can be obtained.

The other problem is related to the multiplicity of the association end at the power type class in the case when the respective power is set with {incomplete} constraint, see Figure 3. Such multiplicity should be set as 0..1, while in the examples presented in

[11], and what is surprising in the UML specification [15], the multiplicity is set to 1. This is not correct because there may exist some objects of the respective superclass that are not members of its subclasses.

References

- [1] Ambler, Scott W.: *Agile Database Techniques. Effective Strategies for the Agile Software Developer.*, John Wiley&Sons, 2004.
- [2] ANSI/ISO/IEC International Standards (IS): Database Language SQL — Part 2: Foundation (SQL/Foundation), 1999, Available at: <http://www.cse.iitb.ac.in/dbms/Data/Papers-Other/SQL1999/ansi-iso-9075-2-1999.pdf>
- [3] Blaha M., Rumbaugh J.: *Object-Oriented Modeling and Design with UML*, Second Edition, Pearson Education, 2005.
- [4] Booch G., Rumbaugh J., Jacobson L.: *The Unified Modeling Language User Guide*. Addison-Wesley, 1999 (tłum. pol., *UML podręcznik użytkownika*, WNT, 2002).
- [5] Connolly T., Begg C.: *Database Systems, A practical Approach to Design, Implementation, and Management*. Addison-Wesley, Fourth Edition, 2005.
- [6] Date C., Darwen H.: *Omówienie standardu języka SQL*, WNT, Warszawa 2000.
- [7] Krutchen P.: *Rational Unified Process. An Introduction*. Addison Wesley Longman Inc., 1999.
- [8] Maciaszek L., Liong B.: *Practical Software Engineering. A Case Study Approach*, Pearson Addison-Wesley, 2005.
- [9] Maciaszek L.: *Requirements Analysis and System Design*. Addison Wesley. Second Edition. 2005.
- [10] Martin, J., Odell James J.: *Podstawy metod obiektowych.*, WNT, Warszawa, 1997.
- [11] Rumbaugh J., Jacobson L., Booch G.: *The Unified Modeling Language. Reference Manual*, Second Edition, Addison-Wesley, 2005.
- [12] Naiburg E. J., Maksimchuk R. A.: *UML for Database Design*, Addison-Wesley, 2001.
- [13] Sparks G.: *Database Modeling in UML, Methods & Tools*, pp.10 – 22, 2001.
- [14] Ullman J. D., Widom J.: *Podstawowy wykład z systemów baz danych*, WNT, Warszawa 2000.
- [15] UML 2.0 Superstructure Specification, *OMG*, September 2003.
- [16] Rational Rose Data Modeler. IBM Rational. *Rational Software Corporation*

Usability of UML Modeling Tools

Anna BOBKOWSKA and Krzysztof RESZKE

Gdańsk University of Technology,

Narutowicza 11/12, 80-952 Gdańsk, Poland

e-mails: annab@eti.pg.gda.pl, krzysztof_reszke@o2.pl

Abstract. Usability aspects of UML modeling tools have impact on efficiency of work and satisfaction of software developers. This paper discusses applicability of usability techniques to UML modeling tools evaluation. It describes an empirical study of performance testing of six UML modeling tools and inquiries regarding usability problems with these tools. Then, Goals, Methods, Operators, Selection Rules (GOMS) method is applied for investigation of the effort necessary to create diagrams with these tools.

Introduction

UML Modeling Tools play an important role in software development and evolution. They are a kind of Computer Aided Software Engineering (CASE) tools and they are expected to reduce amount of work required to produce a work product and present software artifacts at a level of abstraction which allows for effective decision making. As Unified Modeling Language (UML) became de facto standard of specification and construction of software systems, UML modeling tools support precise and modifiable product documentation, effective team communication and software quality improvement. Many benefits are reported in the studies investigating the impact of UML and its tools on software development, e.g. easier access to information about product, new verification possibilities, easier maintenance, easier version management and generation of documents for customers [2], [9]. However, the level of user satisfaction is medium and improvement in productivity is medium as well. The kinds of benefits depend on characteristics of application, such as the level of fit of CASE tools to the system under development, the level of fit to business processes in software company and the level of integration with other CASE tools. But they depend also on the usability of the UML modeling tools.

While tracing evolution of UML modeling tools one can notice the beginnings of the CASE tools in the 1990s [4], [11]. The tools related to modeling were usually kinds of diagram editors for structured or object-oriented methods. At this time, a lot of research on modeling notations was done and many visions of potential benefits related to automatic model processing have appeared, e.g. automatic code generation, simulation and verification. In the 1997, the first version of the UML standard was published and, shortly after that, two leading UML modeling tools were Rational Rose [12] and Select Enterprise. They were expensive, but also more advanced than diagram editors – they had repository which stored the model underlying the diagrams and allowed to generate skeleton of class code and documentation. After the year 2000, many UML modeling tools have appeared on the UML modeling tools market. Almost all of them have repository, check consistency of different diagrams, enable code

generation from class diagram to C++ and Java, generate documentation and allow for a kind of reverse engineering. Some of them additionally offer generation of database from class diagram, transformation to XMI or round-trip engineering. The prices are smaller and a few vendors supply a version for free download.

Since there are many UML modeling tools in the market, some criteria to evaluate them are needed when a company wants to purchase such a tool. An obvious criterion, which is relatively easy to compare, is the cost. The more difficult is evaluation of the technical quality and fit of a tool for company specifics. The list of technical features to be considered includes [8]: repository support, round-trip engineering, HTML documentation, full UML support, versioning, model navigation, printing support, diagrams views, exporting diagrams, scripting, robustness, platform consideration and new releases. In the near future, the next level of sophistication is expected. The list of expected features contains: integrated editor of diagrams and code, generation of code from interaction and state diagrams, integration with metrics and management facilities, technology based on Scalable Vector Graphics, and interoperability with other tools based on XMI. There are also propositions for further evolution: support for UML extensions, integration with business model, fit to MDA technology and ergonomics. The last list is related with recent research activities in the area of UML modeling tools.

Usability (ergonomics, user interface design) issues are concerned with such concepts as ease of learning, ease of use, productivity and user satisfaction. The usability techniques help to eliminate annoying features, artificial and complicated ways of doing things, or features which invite mistakes. The goal is to give a good support for business tasks, so that users can concentrate on their work, and not on the way of using a tool. Usability is related to many technical characteristics, e.g. functionality, stability, robustness, performance. However, it deals with some very specific aspects and delivers techniques to check them, e.g. techniques for understanding and modeling goals of users, their tasks and use of software tools in their context of use. It enables analysis of a tool in context of its application, although basic usability characteristics are important to any kind of software. Although usability is a useful perspective for UML modeling tools evaluation, not much regular research investigating this topic in details has been done.

The goal of this paper is to present an approach to applying usability techniques to evaluation of UML modeling tools. Section 1 discusses the applicability of usability techniques for evaluation of UML modeling tools. Section 2 describes experimental design and conduct of the empirical study with joined use of performance testing and inquiries. In the next two sections, the analysis of quantitative and qualitative data collected in this study is presented. Section 5 briefly discusses application of Goals, Methods, Operators, Selection Rules (GOMS) method and Section 6 – the results of this application. Section 7 integrates results and draws conclusions.

1. Usability Techniques for Evaluation of UML Modeling Tools

While reviewing usability techniques for the use in the area of UML modeling tools evaluation, it is useful to pose the following questions: What are potential benefits of such application? What is the added value comparing to software engineering techniques? Which usability techniques are most useful?

Usability techniques can be classified into analytical and empirical studies. In the analytical studies researchers check out selected features according to a theory, human-

computer interaction (HCI) principles or cognitive principles. The empirical studies require participants who manipulate software in controlled experiments or they are under observation in the case of field studies. Another classification can be made into inquiry, inspection and testing methods. [13]

Inquiry techniques are related with observations or querying users in order to discover the context of use of products, determine usability requirements or learn about experiences and preferences with the product. They include: contextual inquiry, ethnographic study, field observation, interviews, focus groups, surveys, questionnaires, self-reporting logs or screen snapshots.

Usability inspections are more or less formal reviews concerned with usability issues. In heuristic evaluation usability specialists judge whether each element of a user interface follows established usability principles, e.g. "Recognition rather than recall" or "Flexibility and efficiency of use". Cognitive walkthrough is a review technique where expert evaluators construct task scenarios and then play role of users working with that interface. Pluralistic walkthroughs are meetings where users, developers, and usability professionals walk together through task scenarios, discussing and evaluating each element of interaction.

Usability testing is carrying out experiments. The methods include performance measurement, thinking-aloud protocol, question-asking protocol (users are asked direct questions about the product instead of waiting for them to vocalize their thoughts) and co-discovery in which two participants attempt to perform tasks together while being observed. Usability labs contain equipment, which enables use of many methods and integration of the results.

Cognitive modeling produce computational models for how people perform task and solve problems based on psychological principles. A usability technique, which has already been adopted by software engineering, is user interface prototyping.

Traditional software development methods focus on elicitation of functional as well as nonfunctional requirements for software product. The main goal is to develop a system which satisfies needs of all stakeholders who have only their own, usually incomplete, perspective on the system under development. In order to make it, interviews and discussions, observations of potential users and similar systems at work, analysis of the internal documents and formal documents of the domain can be performed. Different kinds of reviews are in use while checking out analysis or design documents for completeness and correctness. The difference between usability and software engineering techniques is focus. While software engineering is concerned with mainly functional and technical quality, usability analyses characteristics associated with the context of use and ease of use.

Case studies, formal experiments and qualitative methods of data collection are used in the empirical software engineering research. They seem to be similar to performance measurement from the usability toolbox. However, despite of the similar basic descriptions, they are different. The phenomena to be explored in software engineering are concerned with software project and its techniques. They measure time of performing them as well as quality characteristics, e.g. number of defects found using different inspection methods. In the usability research the notion under measurement is the time of performing business task using software tools.

No single study answers all usability questions. The techniques to be used in concrete situation depend on the appropriateness of the techniques to answer a research question, the time and the cost of evaluation and other constraints, e.g. available human resources. Empirical studies are necessary to acquire realistic data. Analytical methods

are useful because they are much cheaper. Quantitative and qualitative methods deliver complementary information. The expected benefits of applying usability techniques to UML modeling tools evaluation are possibility to focus on the context of use and to identify features of user interface which would make modeling tools more useful. Making use of them when developing UML modeling tools, should improve productivity and satisfaction of software developers.

2. Experimental Design and Conduct

The goal of the study was to test empirically performance of selected UML modeling tools. The study was based on the assumption that the time of performing tasks is one of the usability indicators. The following research questions were posed: Which modeling tools allow for the fastest modeling? Which tool features are most efficient? The plan included collection of performance data and comments. The participants of the study would be given modeling tasks and they would collect time of performing them as well as comments on features which facilitate modeling or cause unnecessary difficulties. The analysis of results would compare the time needed to perform the tasks while working with different tools and would attempt to find explanations using the comments.

One of the main issues in empirical studies is reliability of results. The problems to overcome are acquiring large enough number of representative participants and managing impact of a number of independent variables which are not the topic of research question. In this study every effort was made to eliminate or, at least, capture and analyze impact of all independent variables which could distort the results. The summary of identified independent variables, explanation of their impact on results and the ways of dealing with this impact are presented in Table 1.

Table 1. Independent variables, their impact and ways of dealing with this impact

Variable	Explanation of the impact	Way of dealing with this impact
System under modeling	Different systems have different complexity and different amount of time is needed to model them.	All participants model the same system.
Modeling language	The same system might be more or less difficult to model depending on the diagrams which are used for modeling.	All participant model with the same types of UML diagrams: use case, class, sequence and state diagrams.
Environment	Parameters of computers used in the study have impact on performance.	All participants worked on the same type of Dell computers with Intel Pentium 4, 1.7 GHz.
Style of thinking	Different participants may need more or less time to elaborate the models and the results could be at different level of abstraction.	Participants do not model, but they are given sheets of paper with diagrams to be re-drawn with a tool.
Knowledge, experience	The performance can differ depending on knowledge and experience of participants.	All participants are students of the same semester with experience with Rational Rose.
Personality features and skills	Dynamism, determination, skills at modeling etc. may impact the time of performing tasks.	Personality features are measured and correlation with time of performing tasks is calculated.

Motivation and attitude to modeling	Motivation and attitude to modeling may impact the time of performing tasks.	All students are motivated to work fast and deliver good comments. The level of personal motivation is measured and correlation is calculated.
-------------------------------------	--	--

The studies were conducted in December 2004 and January 2005. The UML modeling tools under test were: Visual Paradigm for UML Community Edition 4.1 [14], Enterprise Architect 4.5 [3], Jude Community 1.4.3 [5], Meta Mill 4.0 [7], Poseidon for UML Personal Edition 3.0 [10] and ArgoUML v.0.16.1 [1]. The participants were 58 third-year students of Computer Science course at Gdansk University of Technology. They had previously a lecture about UML and modeled a system with Rational Rose [12]. The following numbers of participants have worked with the tools during the empirical study: Visual Paradigm – 13 persons, Enterprise Architect – 8, Poseidon – 10, Jude – 13, ArgoUML – 7 and MetaMill – 7.

After a short introduction, the participants were given four sheets of paper with description of tasks for use case, class, sequence and state diagrams respectively. The description of tasks included instructions on how to perform the tasks (a list of subtasks), a diagram presenting Electronic Class Register, and the tables for reporting start time of performing each task and finish time of all subtasks. On the other sides of the sheets there was a space for comments. An additional sheet was delivered for measuring personality features regarding attitude to modeling, skills at modeling, dynamism, determination, attitude to the study and motivation.

3. Results of the Performance Testing

The main task of the analysis of experimental data was transformation of collected start and finish times for each subtask into average duration time of drawing all diagrams for each UML modeling tool. It included checking for correctness of data, calculation of duration time for each participant and calculation of average time for each tool. Then, the comparison of average duration times for different tools was made and analysis of the correlation between personality features factor and the duration time of drawing all diagrams was calculated. Additionally, analysis of the duration time of drawing particular diagrams was made.

In the first step, all data from the sheets of paper were inserted into spreadsheet tables and checked for correctness. It included elimination of spelling errors, identification and elimination of incomplete, imprecise or difficult to interpret data. Then, calculation of the duration time of drawing a particular diagram (t_d) was made according to Eq. (1) and calculation of the duration time of drawing all diagrams (t_c) – according to Eq. (2).

$$t_d = \sum_{i=1}^n (t_{f_i} - t_{f_{i-1}}) . \quad (1)$$

where

t_d – duration time of drawing a particular diagram by a given participant

n – number of subtasks associated with the diagram

t_{fi} – finish time of subtask i
 t_{p0} – start time of drawing the diagram

$$t_c = \sum_{i=1}^4 t_{di}. \quad (2)$$

where

t_c – duration time of drawing all diagrams by a given participant
 t_{di} – duration time of drawing use case($i = 1$), class($i = 2$), sequence($i = 3$) and state($i = 4$) diagrams

In the next step, the average duration time (t_{cavg}) for each UML modeling tool was calculated according to Eq. (3).

$$t_{cavg} = \sum_{i=1}^n \frac{t_{ci}}{n}. \quad (3)$$

where

t_{cavg} – average duration time for a UML modeling tool
 n – number of participants using selected tool
 t_{ci} – duration time of drawing all diagrams by participant i

The ranking of UML modeling tools was made on the basis of these average duration times. Table 2 presents the tools in the ranking together with the average time of drawing all diagrams and the difference to the best result. The shortest time was achieved for Visual Paradigm. At next two positions are, with a very small difference, Jude Community and Meta Mill. They are followed by two tools implemented in Java environment: ArgoUML and Poseidon for UML. The longest time of drawing diagrams was achieved with Enterprise Architect.

Table 2. Ranking of the UML modeling tools according to the time of performing tasks

UML modeling tool	Average duration time	Difference to the best	
	[hour:min:sec]	[hour:min:sec]	[%]
1. Visual Paradigm CE 4.1	0:50:58	0	0
2. Jude Community 1.4.3	0:56:52	00:05:54	11,58
3. Meta Mill 4.0	0:58:09	00:07:11	14,09
4. ArgoUML v.0.16.1	1:03:18	00:12:20	24,20
5. Poseidon for UML PE 3.0	1:03:33	00:12:35	24,69
6. Enterprise Architect 4.5	1:12:22	00:21:24	41,99

Individual differences between participants could distort the results achieved in the experiment. For example, it could happened that good at modeling, dynamic and highly-motivated participants were in one group and poor, slow and poorly motivated

in others. Analysis of the impact of personality features was done by checking out correlation between the duration time and personality features measures. All of them could have logically contradictory impact, e.g. the more dynamic person, the shorter duration time. Thus, personality feature factor was calculated as an average of personality features for a given participant. There was no clear correlation between the personality features and duration time of completing tasks. The dependency between the average duration time (t_{cavg}) and the personality features factor is presented in Figure 1. It seems to be totally random. The results of this analysis show that there was no correlation between personality features and time of performing tasks, and thus the average times were not distorted by the individual differences.

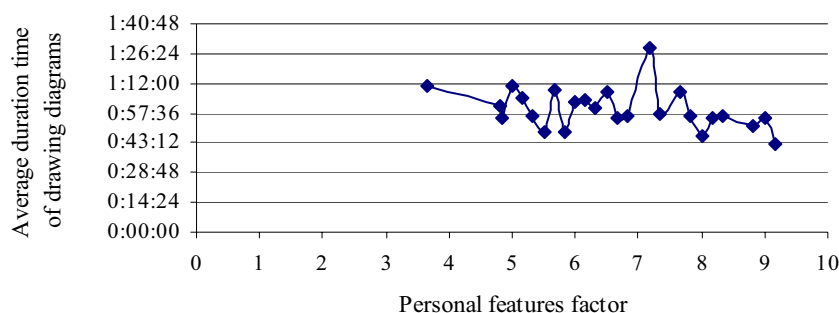


Figure 1. Dependency between personal features factors and average duration time

The comparison of the average duration times for the same type of diagram can indicate ease or difficulties of drawing certain types of diagrams. In fact, the differences are quite large. For example, the time of drawing use case diagram with Enterprise Architect was twice longer than with Jude and the time of drawing sequence diagrams with Visual Paradigm was twice shorter than with Jude and ArgoUML. Although Visual Paradigm is at the first position, the time of drawing use case diagram is 46% more efficient with Jude.

4. Analysis of User's Opinions

During the experiment the participants have reported suggestions, comments and usability problems related to the tools. These data were analyzed for each UML modeling tool in order to identify problems, frequency of their occurrence and their impact on performance. The common problems, which appeared in comments for different tools, are grouped and presented in Table 3.

Table 3. Descriptions of common problems and related tools

Description of problems	Related tools
Improperly functioning UNDO command causes difficulties with removal of objects (they are inactive or they are not entirely deleted)	Visual Paradigm, Poseidon PE, ArgoUML

No keyboard shortcuts or the accelerator-keys for adding new methods, attributes or objects on the diagrams.	Visual Paradigm, Poseidon PE, Enterprise Architect, ArgoUML, Meta Mill
Difficulties with controlling the objects lifeline and labeling messages on sequence diagram (e.g. drawing new messages causes break of this lifeline)	Poseidon PE, Jude Community, Meta Mill
Enter button does not allow to accept object's name but redirects to the next line. In order to accept the name there is a need to click on another object.	Visual Paradigm, Enterprise Architect, ArgoUML
Difficulties with setting up multiplicity of associations (hidden forms) and unintuitive layout of the operations.	Poseidon PE, Enterprise Architect, Jude Community

The positive opinions about Visual Paradigm included ease of adding multiplicity of associations, helpful options in pop-up menus like “Association End From” and “Association End To” and existence of handles around objects giving possibility to add associated elements. The problem occurred when users tried to enter long names of objects (i.e. longer than 20 characters) and noticed that they are not automatically divided into few rows but abbreviated. (Participants have not found a “Fit Size” icon which allows to change it.)

The users of Poseidon PE noticed that a lot of handles around objects caused lack of clarity of diagrams and required high precision while creating any association between objects. Another problem were slow and delayed responses from the system while creating objects, associations and setting up additional parameters (e.g. changing the properties of object, adding documentation).

As an advantage of using Enterprise Architect 4.5 and Meta Mill participants reported additional bookmarks which allowed to switch quickly between objects (e.g. switching between class and sequence diagrams). Among the deficiencies of Enterprise Architect 4.5 they reported unnecessary warnings while typing white spaces in object's name (e.g. ‘a space embedded in object's name may cause code generation problems’) as well as the difficulties with closing the start page and setting up a diagram space.

The positive opinions of users who worked with Jude Community included intuitive and easy to understand icons on diagram's toolbar which allowed to create quickly associations, object and messages and existence of keyboard shortcuts for adding new methods, attributes and objects on diagrams (e.g. CTRL-F to add new operation to classes). The negative comment said that long names of objects (e.g. names longer than 20 characters) were not automatically divided into few rows but abbreviated and users could not scale objects.

The users of ArgoUML noticed that there was a convenient option for scaling diagram's size (e.g. user can scale a diagram in range 0 to 500 % and observe the changes) and they were complaining about the lack of online help.

It is worth to mention that the use of the same mechanism (e.g. handles around object) can be simple and efficient for some tools and difficult for others. The implementation details matter. For example, handles around an object work fine in Visual Paradigm and they cause difficulties in Poseidon.

5. Analysis of UML Modeling Tools with GOMS

GOMS stands for Goals, Operators, Methods and Selection Rules [6]. It is an analytical method which allows to predict and evaluate the usability of programming tools, user interfaces or any complex applications. Goals represent the goals which users try to accomplish and they are decomposed into hierarchy of sub-goals. The leaf sub-goals correspond to methods. The methods consist of elementary steps made with operators. The operators are atomic-level operations, e.g. keystroke, button click or cursor move. Selection Rules are used to decide which method to use when several methods for accomplishing a given goal are applicable.

There are various kinds of the GOMS model. The simplest one is Keystroke-Level model in which predicted execution time is the sum of elementary user's keystrokes. In the most complex CPM-GOMS tasks execution depends on user's perceptual, cognitive and motor processes. Natural GOMS Language (NGOMSL) defines way of describing user's methods in a program-like manner. While applying the GOMS model it is necessary to make tasks analysis and identify user's goals. It is used to check out procedural aspects of usability, e.g. number of steps needed to accomplish a goal, consistency of steps while following different procedures and efficiency of them. GOMS model is practical and effective when the problem under exploration fits above specifics. Otherwise it might be too costly and time-consuming. Some extensions for event-driven applications have been recently developed.

The analysis of UML modeling tools with GOMS consisted of the following steps: creating hierarchy of goals (one for all tools), describing all methods for each UML modeling tool, counting numbers of steps and comparing them among the tools. The main goal in GOMS-for-UML-tools goals hierarchy is creating model of a system. It is decomposed for sub-goals associated with creating several types of diagrams: use case, class, state, and sequence diagrams. The sub-goals for creating a diagram depend on specifics of this diagram, e.g. for use case diagram sub-goals would be: creating diagram space, adding system boundary, adding elements to the diagram space, adding associations between elements and adding documentation. A fragment of specification in the GOMS language (GOMSL) for creating diagram space is presented in Table 4.

Table 4. Specification of method "Create diagram space"

Method_for_goal: Create_Diagram_Space	
Step 1.	Recall_LTM_item_whose Name is "New Diagram" and_store_under <command>
Step 2.	Look_for_object_whose Label is Containing_Menu of <command> and_store_under <target>
Step 3.	Point_to <target>
Step 4.	Click left_mouse_button
Step 5.	Select Type_of_diagram of <current_task> and_store_under <target>
Step 6.	Point_to <target>
Step 7.	Click left_mouse_button
Step 8.	Verify "correct space of diagram was created"
Step 9.	Return_with_goal_accomplished

An addition to original GOMS models was parameterization. Unlike simple commands which are made just once, modeling requires repeating some operations several times, e.g. adding a few classes, then adding a few attributes and operations to classes. The goals which are accomplished more often should have larger impact on the final result than these which are accomplished seldom. The idea of extending GOMS model was to add possibility to set up numbers of using the methods. In the GOMS-for-UML-tools model, the parameters can be set up for prediction in concrete situation. The studies describe in the next section use parameters equal to the numbers of elements in the empirical study.

6. Results of the Analysis with GOMS

The numbers of steps needed to perform sub-goals associated with class diagram and numbers of using them when performing experimental tasks are presented in Table 5. The smallest number of steps while creating diagram space in Poseidon is result of an accelerator-key. While making two objects and their association in most of the tools users must add two objects (AOO) and then add association (AA). An exception is Visual Paradigm, which enables adding associated objects directly (AAO). While adding association (AA) the number of steps depends on existence of association name and the number of adornments. The notation 16/12 means that 16 steps are used to add an association with name and multiplicity and 12 steps – for associations without them (e.g. aggregation, composition). They are used 11 and 3 times respectively. Add Parameter (AP) method is used to add a new attribute or operation to a class. The number of steps in range of 8 to 17 is the results of existence or not of hand menus, accelerator-keys or special bookmarks. It has a significant impact on the number of steps in total because adding parameters is used many times.

Table 5. A fragment of table with numbers of steps for class diagram (Abbreviations: CDS – Create Diagram Space, AOO – Add Object Only, AAO – Add Associated Object, AA – Add Association, AD – Add Documentation, AP – Add Parameter)

Methods for diagram	VP	Jude	Pos.	Argo	Meta	EA	Number of uses	
	The numbers of steps for sub-goals						VP	Other
	Class diagram							
1. CDS	9	9	4	9	9	9	1	1
2. AOO	10	9	9	9	11	10	1	10
3. AAO	11	0	0	0	0	0	14	0
4. AA	16/12	22/17	25/23	25/23	22/19	22/20	11/3	11/3
5. AD	9	9	9	10	9	8	10	10
6. AP	10	8	13	13	15	17	37	37

The ranking of UML modeling tools and total amount of steps needed to perform the tasks are presented in Table 6. The smallest number of steps needed to accomplish the main goal was required for Visual Paradigm. At the second position is Jude

Community with the result of only 16% of steps more than the leading tool. At next two positions are Poseidon and ArgoUML with very similar results. The numbers of steps needed for these tools are larger more than 25% comparing to Visual Paradigm. The next tool is Meta Mill, the only big difference in comparison with Table 2 (Ranking of UML modeling tools according to the time of performing tasks). At the last position is Enterprise Architect. Although one could argue that some steps might be more difficult and time consuming than others, it seems that GOMS model at this level of abstraction enables quite good predictions. It confirms empirical results. An attempt to model the level of difficulty and the amount of time needed to perform basic operations would be difficult to capture precisely, time-consuming and prone to errors related to individual differences.

Table 6. Ranking of the UML modeling tools according to number of steps using GOMS

UML Modeling Tool	Number of steps	Difference to the best result	
		[steps]	[%]
1. Visual Paradigm CE 4.1	1428	0	0
2. Jude Community 1.4.3	1658	230	16,11
3. Poseidon for UML PE 3.0	1794	366	25,63
4. ArgoUML v.0.16.1	1837	409	28,64
5. Meta Mill 4.0	1959	531	37,18
6. Enterprise Architect 4.5	2006	578	40,48

7. Conclusions

This paper has presented the approach to application of usability techniques to evaluation of UML modeling tools and the comparison of six UML modeling tools. The empirical studies included performance testing and querying participants about usability problems. Then, the analytical method GOMS was applied for modeling users goals and comparing the numbers of steps needed to accomplish the goals with each tool. With both empirical and analytical methods rankings of UML modeling tools were made. With only one exception the results were confirmed. Starting from the best, they are: Visual Paradigm, Jude, then Poseidon and ArgoUML with a very small difference and Enterprise Architect. Metamill was at the third position in the ranking from empirical studies, and at the fifth in the ranking using GOMS.

The studies confirm that usability issues are important. A lot of work remains to be done in this area. In order to assure usability of UML modeling tools, their developers should make an effort to apply some usability techniques. Different usability methods deliver complementary results. Performance testing delivers performance data from participants working with the tools. GOMS method is a cheap methods for predicting performance. Usability guidelines facilitate reviews by delivering usability principles, and user's opinions allow to find out strengths and weaknesses of the solution. It is worth to notice that two leading tools in the study had a kind of smart solutions (e.g. associated objects, shortcuts, bookmarks, hand-menus) which enabled better performance. Although usability testing is responsibility of software companies which release UML modeling tools, there is a job for researchers as well. They can identify efficient

solutions, indicate annoying features in current technology and work out a list of usability guidelines for UML modeling tools.

References

- [1] ArgoUML, <http://argouml.tigris.org/>
- [2] Bobkowska A.: Efektywność modelowania systemów z perspektywy praktyki w firmach informatycznych. In: Efektywność zastosowań systemów informatycznych. *WNT* (2004)
- [3] Enterprise Architect, <http://www.sparxsystems.com.au/ea.htm>
- [4] Fisher A.S.: CASE Using Software Development Tools, *John Wiley & Sons* (1991)
- [5] A Java/UML Object-Oriented Design Tool [JUDE], <http://jude.esm.jp/>
- [6] Kieras D.: A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN3. [ftp.eecs.umich.edu people/kieras](ftp.eecs.umich.edu/people/kieras) (1999)
- [7] Metamill (tm) – the Visual UML (tm) CASE Tool, <http://www.metamill.com/>
- [8] Objects by Design, <http://www.objectsbydesign.com>
- [9] Piechówka M., Szejko S.: CASE, jaki jest, każdy widzi. *Informatyka 4* (1998)
- [10] Poseidon for UML, www.gentleware.com
- [11] Pressman R.: Software Engineering. A Practitioners Approach. McGraw-Hill (2000)
- [12] Rational Rose, www-306.ibm.com/software/rational/
- [13] The Usability Methods Toolbox, <http://jthom.best.vwh.net/usability/>
- [14] Visual Paradigm for UML, www.visual-paradigm.com

On some Problems with Modelling of Exceptions in UML

Radosław KLIMEK, Paweł SKRZYŃSKI and Michał TUREK
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-095 Kraków, Poland
e-mails: {rklimek, skrzynia, mitu}@agh.edu.pl

Abstract. During the last couple of years UML has been becoming more popular and has been rapidly accepted as the standard modelling language for specifying software and system architectures. Despite this popularity using UML 1.x caused some trouble and developers face some limitations due to the fact that software industry has evolved considerably during last seven years and software system become more and more complex. One of the most important issues in modern software systems is exception handling. Exception handling mechanism enables to make software systems robust and reliable [6]. However mechanism provided even by the 2.0 version of UML, in authors opinion, seems to be insufficient and is far from well-known implementation practise. UML does not put proper emphasis on exceptions handling the aim of the paper is to present the possibilities of exceptions modelling in UML using statechart diagrams. These approach seems to be the most natural way to model the mechanism. Alternative methods, which should be taken into consideration in future research, are discussed at the end of the paper. UML dialect used is proposed by Telelogic so does the tool Tau G2.

Keywords. Exceptions, interrupts, exception handling, software robustness, UML 2.0, statechart diagrams, Telelogic Tau G2.

Introduction

Exceptions and handling of exceptions are well-known issues during modelling software systems. The typical model of a system consists of many flows of computations, many complex subsystems' interactions and communications, error detections, reports, etc. It often leads to a kind of confusing software spaghetti. Moreover, modern systems have to include many different situations which could be called exceptional ones. These situations stay in the opposition to a normal program execution. However, these anomalies cannot be recognized as typical run-time errors. It is really hard to imagine a modern and complex software system without exception handling. Exceptions are specially important when modelling safety critical systems, real-time systems or, generally speaking, the systems which must always keep the run time executing until their normal and safe termination.

The growing complexity of computer-based systems has resulted in a raise of the level of abstraction at which such systems are developed. Nowadays UML is a very important and most-used tool for the modern system analysis and modelling. It usually makes a good impression to call it a new *lingua franca* for computer science and software engineering. UML has a very strong position in the world of system analysts and developers. There are many interesting tools that support the process of system

modelling and forward engineering. Perhaps one of the most powerful tools for system modelling and verification is Telelogic Tau Generation2 [5]. This is a very mature system. It enables generating a really rich code from almost all UML diagrams including different and complex scenarios and modules interactions. Last but not the least, Tau Generation2 is provided with a special module which enables to simulate and to verify developed software models. This is a very strong and powerful feature of Tau Generation2 which distinguish this system among the others.

The aim of the paper is both to present the possibilities of exceptions' modelling in UML and to examine this process for Telelogic Tau Generation2. It seems that the question of exceptions and handling of exceptions in UML is handled rather in a general manner. Exceptions were discussed secondarily in the former version of UML. It seems that it was far from the well-known implementation practice. A kind of progress is made in UML 2.0 [7] but is still not close enough to the practice of exception mechanism which is implemented in many programming languages. Exceptions are powerful addition to these languages and that is why the exception mechanism of the modern programming languages is an important background for the ideas presented in this paper. It is worth examining how exceptions could be modelled in Telelogic Tau G2 which is very mature and valuable tool for UML. In the context of the problem of exceptions, there is also a necessity to focus our attention on the UML dialect which is proposed by Telelogic.

The rest of the paper is organized in the following way. In the next section, the mechanism of exceptions is discussed. Next, some possibilities of exceptions' modelling in UML are presented, too. In succeeding section, the practical aspects of modelling of exceptions in Telelogic Tau Generation2 are discussed. The conclusions and directions for future work are outlined in the last section.

1. Mechanism of Exceptions

The mechanism of exceptions is a well-known issue in the modern software development. It is used successfully in many programming languages, e.g. [3], though there are some differences among particular implementations of exceptions. Now, it is hard to imagine advanced system modelling without developing exceptions and handling of exceptions. Many advanced topics of exception handling are discussed in literature within the field, see, for example, [2].

An *exception* is a state when occurs a run-time event during a typical flow of instructions which disturbs the normal execution of a program. This event is rather a run-time error however it is classified only as a non-typical and unwanted event. Exceptions allow developers to separate error handling code from normal code. The "normal" code is the most likely code which is to be executed. The error handling code is not expected to be executed often. Perhaps it will be never executed. On the other hand we cannot exclude the bad event we mention above. When modelling exception one have to include into the process of a system development the following, sample and typical questions: what happens if the file is closed? What happens if the stack is overflow? What happens if there is no network connection? etc. There are many classes of computer systems that cannot be suspended or terminated even when the bad event occurs. The exceptions which belong to these systems must always be handled by special procedures.

Generating an exception is called *throwing an exception*. Exception handling requires to break the normal flow of computation. Even though the flow was planned, it must be interrupted in order to carry focus of control to the higher level computations which must handle the generated exception. When the exception handling procedure is found, the exception is handled. The exception handling block is usually placed directly next to the area where exception could occur, i.e. where exception could be thrown. On the other hand, it may happen that an exception could not be accepted by any handler which belongs to this block. In this case the control leaves the current exception handling block and it is *propagated* to the caller, i.e. to the other higher level computations' block which is expected to handle that exception. When the exception is accepted, it is immediately handled. The exception is accepted if the type of the *exception object* which was generated by exception matches the type that can be handled by the handler. The exception object contains information about the bad event, i.e. its type and the state of the program when the event occurred. When the exception is accepted by an appropriate handler it is said that the exception is *caught*. However, the process of exception propagation can be carried on many times. When looking for an appropriate handler, the system stack is decreased. This process is called the *stack unwinding*. The stack is unwound as long as the generated exception is caught. When no handler is found in the whole system, the stack is completely unwound. It means that, if the exception is not caught, the program must terminate. This exception is called the *uncaught exception*. Figure 1 shows a typical program execution and the process of looking for an appropriate handler.

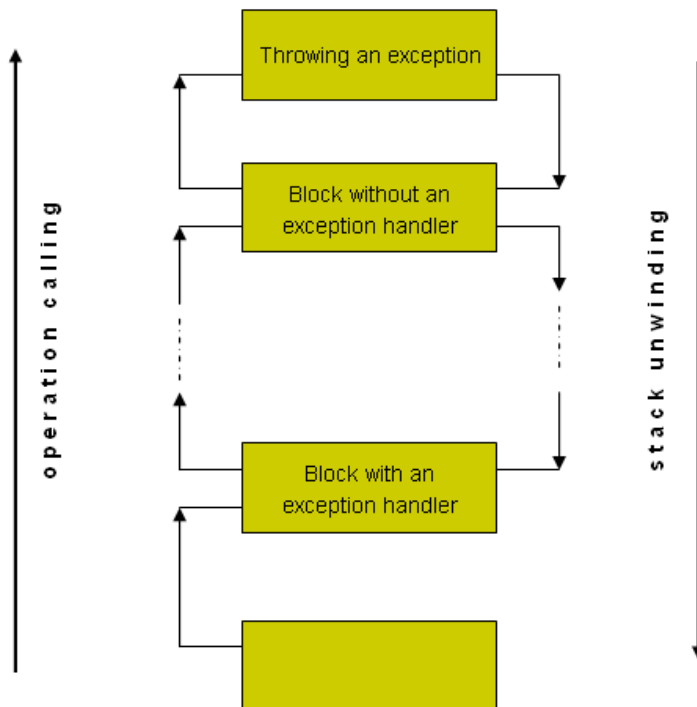


Figure 1. Program execution, creating an exception and looking for a handler

Exception mechanism is well-known in many programming languages, for example C++, Ada95 or Java. Exceptions are usually generated in a special block of code. Let us call it a try block. An exception is created inside a try block by a throw instruction. When the throw instruction generates an exception object, the object is thrown outside the try block. Exceptions are caught inside a special block of code. Let us call it a catch block. The catch blocks usually follow by a try block. The catch block can catch an exception by providing a handler for that type of exception. When an exception is not caught and it leaves a procedure, it is propagated to the other catch blocks. Some programming languages enable to use finally block. When an exception is thrown and the run-time system must leave the procedure, the finally block is executed. It enables to execute some finally operations, for example to release the memory which was allocated inside the procedure. In practise, the finally block enables to differentiate two situations. Firstly, when the normal return from a procedure occurs. Secondly, when an exception is thrown and it is need to break computation and to leave a procedure in an extraordinary way. Figure 2 shows a sample of pair of try and catch blocks.

```

try
{
    instructions;
    throw ...;
    instructions;
    throw ...;
}
catch (...)
{
    instructions;
}
catch (...)
{
    instructions;
}
finally
{
    instructions;
}

```

Figure 2. A sample of pair of try and catch blocks

2. Exceptions and UML

Exceptions are discussed secondarily in UML [1], [4]. It seems that it is far from the well-known practice of modern programming languages.

Exception could be recognized as a system signal. This is a very specific signal which belongs to a class of internal signals. With a signal like that, a special kind of internal communication occurs. Signal is sent (thrown) by an object and it is received (caught) by another object. That kind of communication is represented in Figure 3 as a dependency stereotyped as «send».



Figure 3. Signal and send dependency

As exception is a kind of signal and signal is a special kind of class, thus exceptions could create a generalization relation and inheritance mechanism. When modelling exceptions signals which are sent between active object are considered. A hierarchy of exceptions is shown in Figure 4.

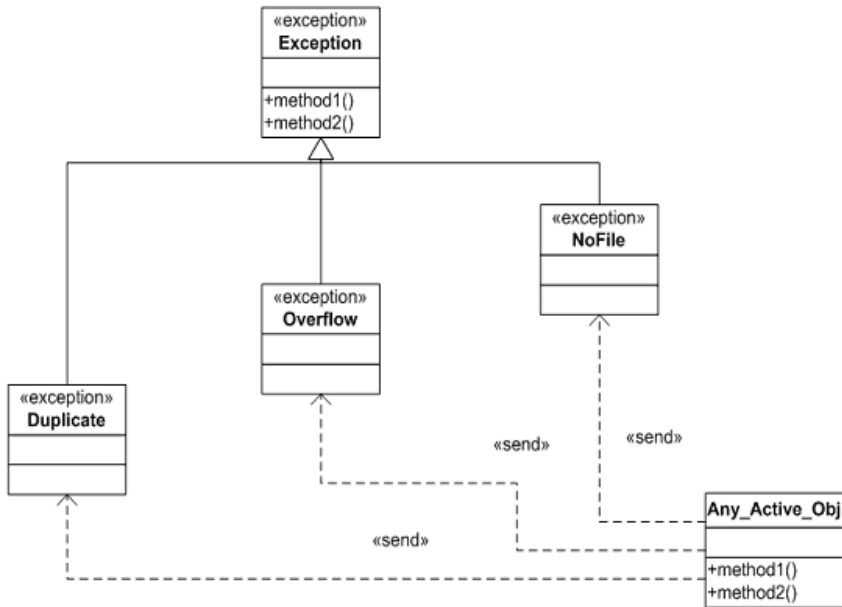


Figure 4. Exceptions and hierarchy of exceptions

Modelling behaviour of exceptions is more difficult to compare with the static aspect of exceptions. Two diagrams seem to be the most convenient for this work. Firstly, state diagrams and, secondly, activity diagrams.

State diagrams describe the states of a system, i.e. all of the possible situations of an object. Each diagram usually represents objects and describes the different states of the object through the system. External or internal events result in states' changing. The event may be an exceptional one, i.e. it may be used for exception modelling. During several UML versions (and dialects) state chart diagrams conventions have been modified. New elements have been continuously added, such as complex decision symbols or signal communication elements. This improves abilities of exceptions modelling in state chart diagrams. Also when modelling exceptions, ports and different kinds of signal interfaces should be considered.

Activity diagrams cover different view of a system. Activity diagrams enable to study the workflow behaviour of a system, i.e. the flow of activities through the system. That is why activity diagrams seems to be a very good candidate for exception modelling. However, activity diagrams will be the next step in our research and these kind of diagrams is not discussed here.

3. Modelling of Exceptions

One method of handling exceptions in UML (Tau) is based on state charts. Using this approach may cause one minor problem that must be solved – receiving signal is acceptable only by communication port. Hence catching an exception is equivalent to receiving a signal, which parameter is an instance of a class which extends an Exception super class. Therefore the class receiving signal must be an active class because state charts can be added in Tau Generation2 only to active classes. Furthermore this class (lets call this class A) must have a communication port to class which method is invoked (lets call this class B). If so class B must also have a communication port to class A in order to enable communication. The port of class B is used to send a signal which is carrying information about exception, which occurred while port of class A is used to receive this signal. In this approach we have to think about “future users” (programmers and software developers who wants to use our class) of designed class. Although this approach is not what we have used to using programming languages it seems that it provides simple and elegant way to solve the problem of handling exceptions in UML.

Solution proposed here is: each class that “wants” to invoke methods of other classes which may throw an exception should be encapsulated by another class (the same in each case) which has a communication port to announce the exception occurrence. This can be modelled on class diagram as generalization-specialization (designed class extends the super class which has required port) or composition (designed class is a part of the class which has required port) relationship between those two classes. In the same time each class which has a method or methods which can throw an exception must be encapsulated by an analogical class. This class has a port which allows to send an exception to announce the fact that exception occurred. This approach is described in Figure 5 – each class that handles exceptions should extend `CatchExceptionClass` (in the example below it is `CatchExampleClass`), whereas each class that has method/methods throwing exceptions should extend `ThrowExceptionClass` (`ThrowExampleClass` in the example). Instances of these classes communicate with each other by sending/receiving signal `ExceptionSignal` which is a part of `Throwable` interface.

As mentioned before solution proposed is based on sending/receiving signal which is carrying information about exception which occurred. Hence this signal should have one parameter – instance of a class that represents an exception – `Exception` class and its subclass all indicates conditions that a reasonable application might want to catch, Figure 6.

Modelling of sample handling in example class `ThrowExampleClass` could be realized by adding state chart to method `method(Integer)` in Tau. Example method throwing an exception is `method` (cf. Figure 7).

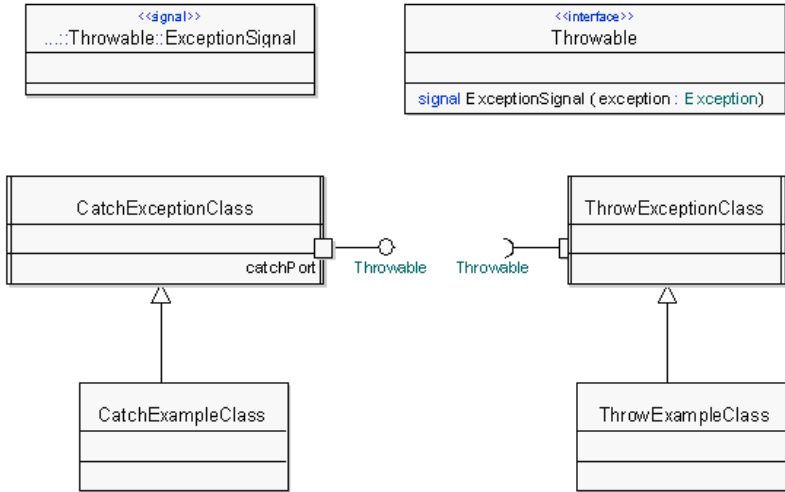


Figure 5. Classes, interfaces signals realizing exception handling

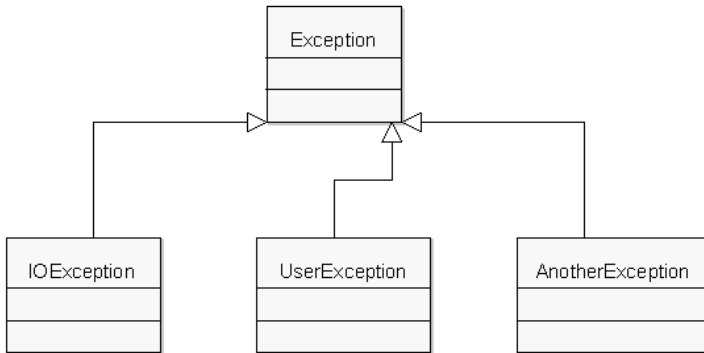


Figure 6. Part of class hierarchy responsible for handling exceptions

The concept of the example is: when invoking `method(Integer)` we pass integer parameter. If the parameter value is below zero exception is thrown – signal carrying exception information is sent. The diagram above is easy to understand however in terms of Tau is not proper. Transition between two states can be activated only by receiving signal. If so some signal should be received between states *Initialize* and *Return*. Unfortunately we do not know what kind of signal, who/what should send it. and we can not provide such a signal if we want to keep state chart clear. This is the only disadvantage of this approach, which solution should be taken into consideration in future research. In approach described (based on state chart diagrams) behaviour of the class which handles the exception may be modelled as described in Figure 8.

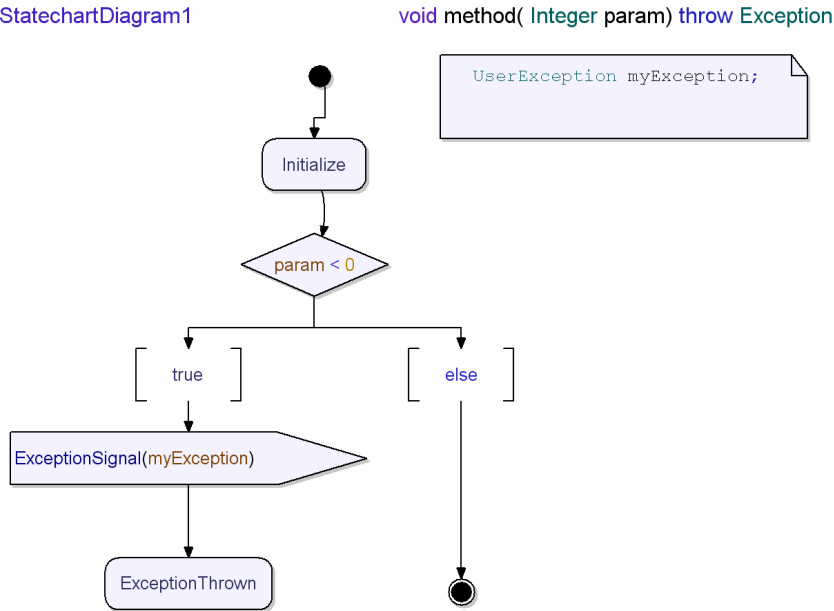


Figure 7. Throwing an exception - example state chart diagram

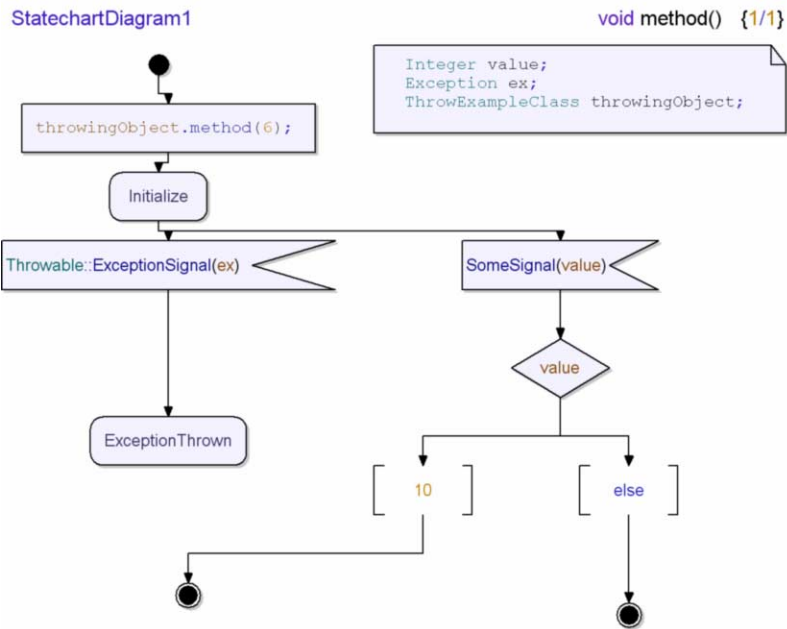


Figure 8. Handling exception - example state chart diagram

Simulating try-catch block which is well known in programming languages, such as Java or C++, is being performed by receiving signal `ExceptionSignal`. The only

disadvantage is that we can not emphasise that the signal can be received only if we invoke method of class `ThrowExceptionClass` before. Unlike previous diagram this diagram is syntactically and semantically correct.

Another approach to the problem could be modelling of whole exception-catching structure inside only one state-machine object. In this way we can bypass the problem around executing correct exception handling operations.

The block of exception catching could be defined here as:

- a block of operations, also with calling another external operations,
- a state machine implementation inside another state (nested states). This structure can be created in the Telelogic Tau with utilisation of “Internal state machines”.

We can also use standard history-states to memorise an exit point, after an exception has been thrown.

The hierarchy of exceptions could be directly passed to hierarchy of class interfaces, deriving from `Throwable`. One of these interfaces will be implemented by a class, which uses concrete type of exceptions. See example in Figure 9.

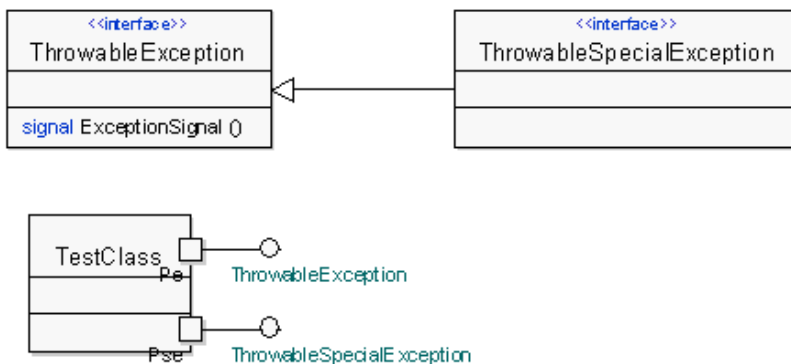


Figure 9. Throwable interfaces and exception signal ports modelling – example class diagram

The problem, which need attention is how to call correct exception handling operation – depending of an exception type. That operation could be placed on any other operation body, when the present operation was called from, according calling level. It is necessary do design a special construction, and use it on each level of sub-operation calling. This structure will consider calling operation return status, and depending of the result of, it will perform one of the following actions:

- it will continue execution after returning of called operation block (will execute finally block or will go on with further code execution when no finally block),
- it will process an exception handling,
- it will pass an exception handling outside when an exception type is incorrect (does not fit to a catching one).

For modelling of presented activity we can use an interface object linked to another interface objects thanks to generalisation/specialisation relation. Each interface will be equipped with exception signals. On the diagram in Figure 10 each exception thrown inside `method2 ()` is being considered outside. `method2 ()` can also include her own exception consideration mechanism.

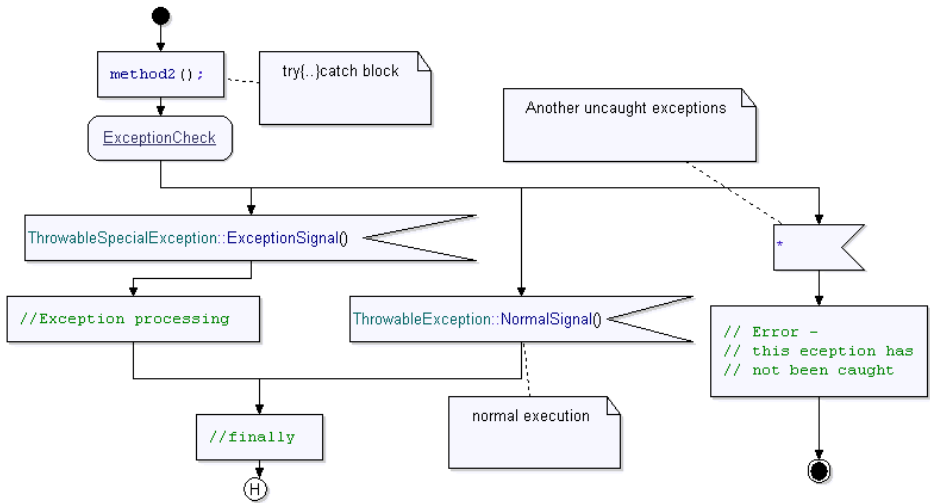


Figure 10. Internal exception consideration mechanism modelling

All exceptions being generated inside internal state `ExceptionBracket1` (see Figure 11) and not caught will be redirected into calling operation code. The implementation of `ExceptionBracket1` could be set as shown in Figure 12.

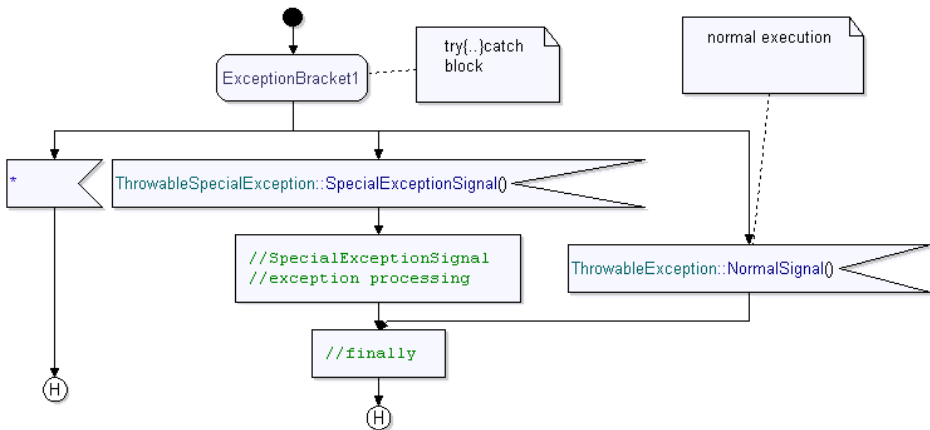


Figure 11. External exception catching brackets modelling

Thanks to usage of this notation we can recognise types of an exception caught – on each level of call-stack. Together with ability of that recognition we can redirect uncaught exception processing to another operation, which the current operation has been called from. In addition we can implement in the model a programming language “finally” block – also on each level.

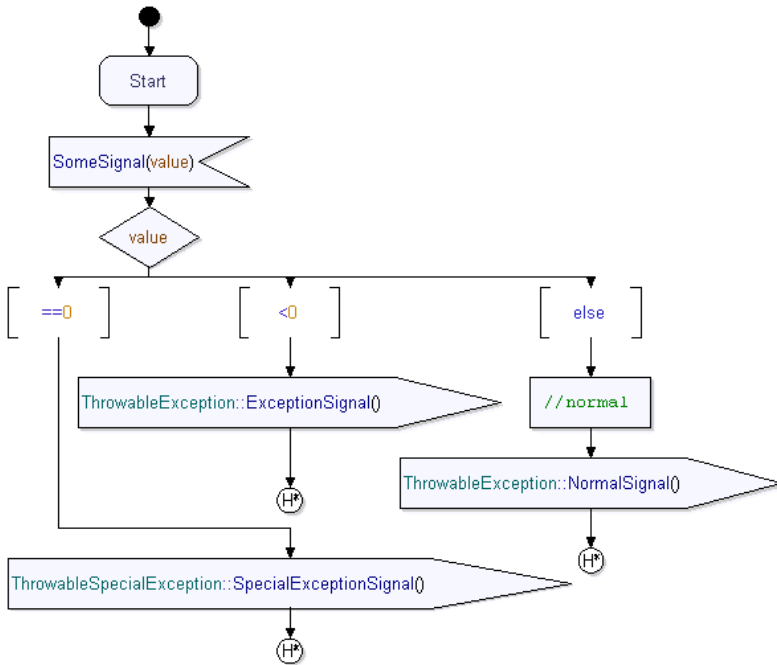


Figure 12. Modelling an exception throwing method

4. Conclusions

Exceptions and handling of exceptions in UML are discussed in this paper. The question of exceptions is very important in computer-based systems. However exception handling is a complicating factor in the design of these systems. Writing programs that include complete and proper exception handling is much more difficult (two to three times [4]) than writing code that is only supposed to work.

Exception handling is not discussed in UML widely. State diagrams and activity diagrams seem to be appropriate tools for exception modelling. State diagrams are examined in these paper. The research is expected to be continued. Activity diagrams will be the next step in the research, too.

Modelling exception handling in UML using approach based on state charts causes some minor problems but give easy to understand and elegant solution. In current version of Tau Generation2 we can precisely describe exceptions thrown by single method but:

- we are unable to model when exception is thrown,
- how to handle the exception in operation body.

In this paper we described approach based on state charts which solves these problems. The only disadvantage of the approach is that in one case state chart diagram which is linked to operation is not correct in terms of Tau. This cause that part of the model which uses the approach is not possible to simulate which is important feature of Tau. Solving this problem will be taken into consideration in future research.

In the alternative method we can base on distinguishing exception type carried by signal ignoring class hierarchy for exceptions which was proposed earlier.

The alternate way of solving problem of handling exceptions in UML is approach based on activity diagrams. Unfortunately in current version of Tau it is not possible to add such a diagram to operation. We are sure that such feature is very important and will soon be added to Tau to enable possibility of modelling handling exceptions. This is also a part of our future research as activity diagrams give easy and elegant way to design operation body.

References

- [1] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, *Addison-Wesley*, 1998.
- [2] PA. Buhr, W. Y.R. Mok, Advanced exception handling mechanisms, *IEEE Transactions on Software Engineering* 26(9) (2000), 820-836.
- [3] M. Campione, K. Walrath, A. Huml, Java Tutorial. A Short Course on the Basics, *Sun Microsystems*, 2001.
- [4] B. P. Douglass, *Real-Time UML*. Developing Efficient Objects for Embedded Systems, *Addison-Wesley*, 1998.
- [5] R. Klimek, P. Skrzyński, M. Turek, *UML i Telelogic Tau Generation2*. Materiały dydaktyczne (in Polish), 2005.
- [6] D. E. Perry, A. Romanovsky, A. Tripathi, Current trends in exception handling, *IEEE Transactions on Software Engineering* 26(9) (2000), 817-819.
- [7] Unified Modeling Language: Superstructure version 2.0, 3rd revised submission to OMG RFP, 2003.

The ISMS Business Environment Elaboration Using a UML Approach

Andrzej BIAŁAS

Institute of Control Systems, Długa 1-3, 41-506 Chorzów, Poland

e-mail: abialas@iss.pl

Abstract. The paper deals with software development for supporting information security management, particularly the modeling of the business environment of the organization implementing Information Security Management System (ISMS). The ISMS, based on the PDCA (*Plan-Do-Check-Act*) model, was defined in the BS7799-2:2002 standard. The paper focuses on the identification of the ISMS business environment that provides appropriate positioning of the ISMS within the organization. The general model of the ISMS business environment based on the high-level risk analysis was presented. The UML approach allows to implement the ISMS within organizations in a more consistent and efficient way and to create supporting tools improving information security management.

Introduction

The development of software supporting the information security management requires specific means and methods, among which the most promising one is the UML-based approach. The Unified Modeling Language (UML) [1], [2] offers different unprecedented opportunities (concerning cost, time, efficiency, etc.) to the system developers in many areas of human activity, like: information technology, business management, logistics, transportation, telecommunications, to solve their specific modeling and design problems.

The paper presents an idea of using a UML-based approach for modeling of the Information Security Management System (ISMS), defined in the BS7799-2:2002 [3] standard, as *“the part of the overall management system, based on a business risk approach, to establish, implement, operate, monitor, review, maintain and improve information security”* within the organization.

The ISMS based on the PDCA (*Plan-Do-Check-Act*) management model, as stated above, should be created on the basis of a business risk approach, and be refined in accordance with the other management systems (business, environment, quality, etc.) coexisting within the organization. This standard provides only general directions how to establish and maintain the ISMS within the organization. Every organization needs more detailed implementation methodology, basing on a wider group of standards and know-how. The implementation of an ISMS, unique for every organization, depends on the organization's business requirements, its environment features and the related risk that hampers the achievement of business objectives.

Identifying and making the description (i.e. elaborating) of the business environment of the organization is not easy and may be a source of discrepancies influencing hidden security gaps, exceeding the cost of implementation and the whole efficiency of the ISMS in the organization.

To solve this problem the UML approach was proposed. The paper deals with a part of a larger work concerning the UML-based IT security management implementations [4], [5]. Its main goal was to research if the UML approach can be more effective than textual description of the ISMS processes, what benefits can be achieved, how to make use of UML elements, what UML features can be most useful, what, and how deeply, IT security management processes can be computer-aided. The elaborated ISMS processes specifications, validated on the large PKI application, have been implemented within the tool prototype that is evaluated by the case studies method. The motivation is that the UML is a widespread specification language and can be understood not only by IT specialists, but also by managers and other specialists.

This is not the first UML implementation within the domain of the security systems development. The works dealing with UML extension, called UMLsec [6], provide a unified approach to security features description. These works are focused on modeling the IT security features and behaviors within the system during development, but not on the IT security management process. Some works are focused on modeling the complex security-related products, consisting of other security-related products, evaluated or not, requiring advanced composition methods. The paper [7] presents an approach, being an extension of Common Criteria (ISO/IEC 15408) [8], called Engineered Composition (EC), focused on the effective composability. The main goal of this work was to achieve the ability to predict that a composition of components will have the desired properties without unpredictable side effects. The UML specification of the generics and components libraries used in computer-aided Common Criteria IT development process was discussed in [9]. There are also papers [10] focused on the UML-based development method, constraints languages and tools used for advanced Java smartcards designs meeting the highest evaluation assurance levels (i.e. EALs) defined by [8].

The paper presents a general idea of using the UML in the ISMS modeling and development, and on this basis it focuses on the introductory stage – the elaboration of the business environment. Business environment elaboration is not explicitly stated in the standard [3] but recommended, preceding the whole PDCA processes and documents development. Usually, implementation methodologies of the ISMS are not very often published in details and they are the consultants' know-how. Examples of the ISMS implementation methodologies, including simple risk analysis methods, are presented in [11], [12]. The paper presents an extended version of the both mentioned, developed on the basis of [13], [14].

The paper presents general structure of the ISMS and its business environment as the UML class diagrams and also business environment elaboration process, using the UML activity diagrams. The paper focuses then only on the initial, but very important, ISMS development phase. Using this concept, the prototype of a computer-aided tool was created, the case studies were performed and the first experiences with it were gathered.

1. A General Scheme of the ISMS and its PDCA Elements as the UML Class Diagram

Basing on the [3], the following general PDCA model was created (Figure 1). The entire ISMS consists of four stages: Plan, Do, Check, Act, each containing elements defined by the standard. All of them were represented by classes grouped in compo-

sitions. Every class represents ISMS documents, like: ISMS policy, risk treatment plan, training programs, incident management procedures, or operations, like: risk analysis tools, security scanners. In reality, the classes shown in the Figure 1, represent a set of classes grouped in hierarchical complex structures.

Please note that the Figure 1 contains one additional class, called *Business environment*, representing ISMS entry into business processes, on which this paper is focused. It is responsible for appropriate positioning of all ISMS elements to meet business needs of the organization. The Figure 1 has first and foremost a conceptual meaning and was developed in details during the project. Starting from the *ISMS scope*, going clockwise to the *Improvement effectiveness*, all ISMS elements specified in [3], can be met.

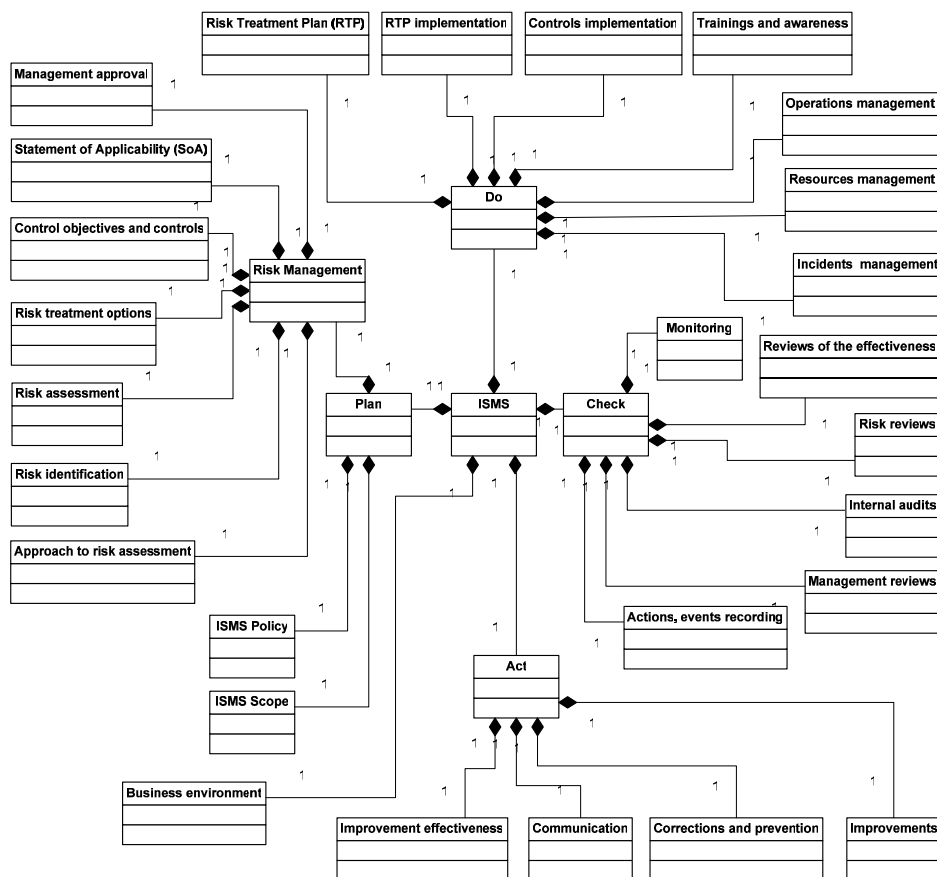


Figure 1. General structure of the ISMS business environment

2. Business Environment Elements and their Elaboration

Because the ISMS is a part of the overall management system of the organization and must reflect its business needs and existing risks, all relationships between information

security and business processes should be identified. The standard [3] focuses on this topic, but does not specify in details how this can be achieved. This identification ensures the appropriate directions of the ISMS development within the organization.

The *Business environment* class is responsible for step by step identification and specification of business needs, risks and working environment of the organization, including the existing quality, environment and other management systems requirements. Performing these works, known as high-level risk analysis, all input information for the Plan stage is provided.

The ISMS business environment is viewed by the set of business level security objectives, expressing security needs within the business domains.

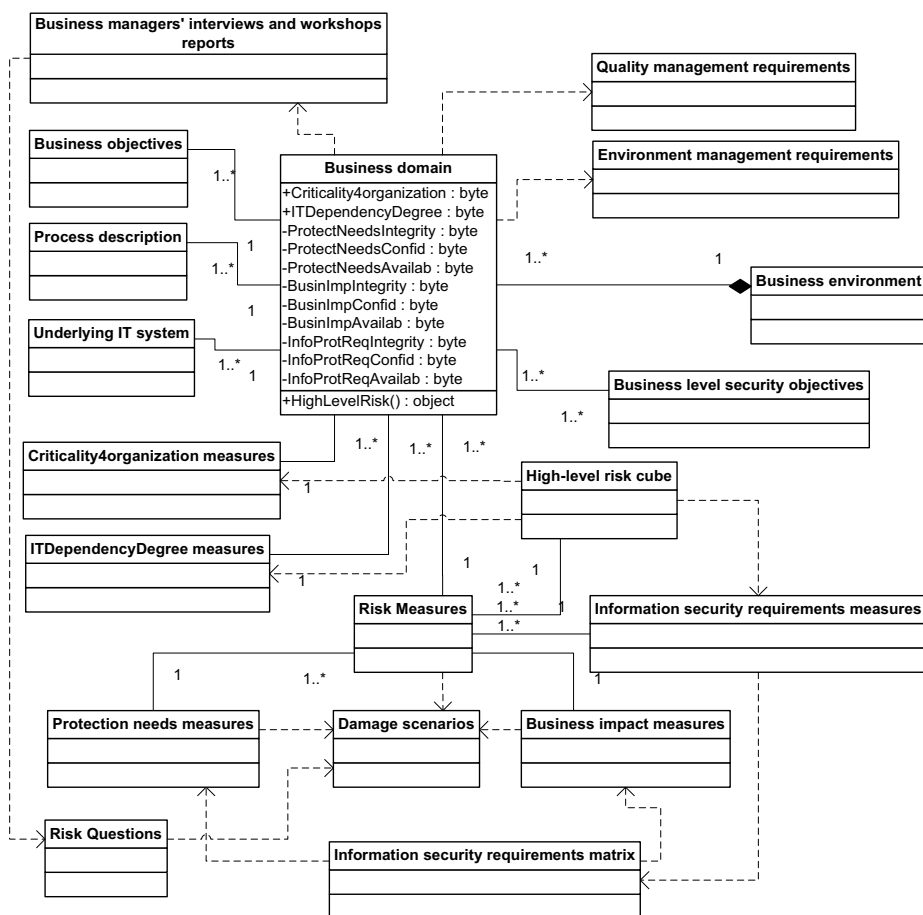


Figure 2. General structure of the ISMS business environment

The central point of the *Business environment* class diagram, presented in the Figure 2, is the *Business domain* class, responsible for specifying a group of well defined business processes, implemented to achieve one or more business objectives within each domain. For every organization it will be possible to distinguish a few

groups of business domains – external, generating incomes, providing business services or services for the society, or – internal, like human resources management, accounting, CRM, etc. The mission of the organization can be decomposed into the sets of business domains. Every domain has its *Processes description* and *Underlying IT systems description*, represented by classes and their subclasses. Usually, the ISMS coexists with other management systems in the organization, represented here by *Quality management requirements*, *Environment management requirement* classes and the like. These requirements, expressed by these classes, should be taken into consideration while creating the ISMS for the organization.

The business domains are identified during business documents review and business managers' interviews and workshops, represented by a specially distinguished class, responsible for sampling and analyzing business information essential to the information security. The business environment elaboration is based on a high-level risk analysis concept [13], and for these purposes a set of quality measures was defined. Each of them encompasses some of the predefined values, like: low, moderate, high, having well defined meaning.

The high-level risk analysis is performed on the basis of a set of *Damage scenarios*. Different types of damage scenarios and *Risk questions* adjacent to them can be taken into consideration, but in this paper well known mechanisms from [14] were adopted:

- violation of laws, regulations or contracts,
- impairment of informational self-determination,
- physical injury,
- impaired performance of duties,
- negative effects on external relationships and
- financial consequences.

Different methods of global risk value derivation [13] can be used (matrices, products), but one of them – the matrix with predefined values, was implemented there as the example.

The development process of the business environment was shown in the Figure 3. All business processes ensuring business objectives should be analyzed and their importance for the organization, as the criticality level, should be assessed. Let us assume (as an example) a simple four-level *Criticality for Organization measure*, as:

- *Very high* – The process encompasses operations, based on reliable information and influencing strongly the organization's survival or the catastrophes for its environment,
- *High* – The process encompasses time-sensitive operations, based on reliable information and influencing the global business position of the organization and its environment (the mainstream processes of the organization's business),
- *Basic* – The process encompasses operations for minor business objectives or supporting mainstream processes, does not influence directly the organization's survival, operations of moderate time- or money-sensitivity,
- *Low* – The supportive process, usually providing internal services, is not time- or money-sensitive, but is needed by the organization.

The other question is the level of participation of IT systems in the realization. This is measured in IT Dependency Degree. Usually, for a modern organization, the most critical processes have the highest value of this parameter. Let us assume (as an example) a simple three-level measure, as:

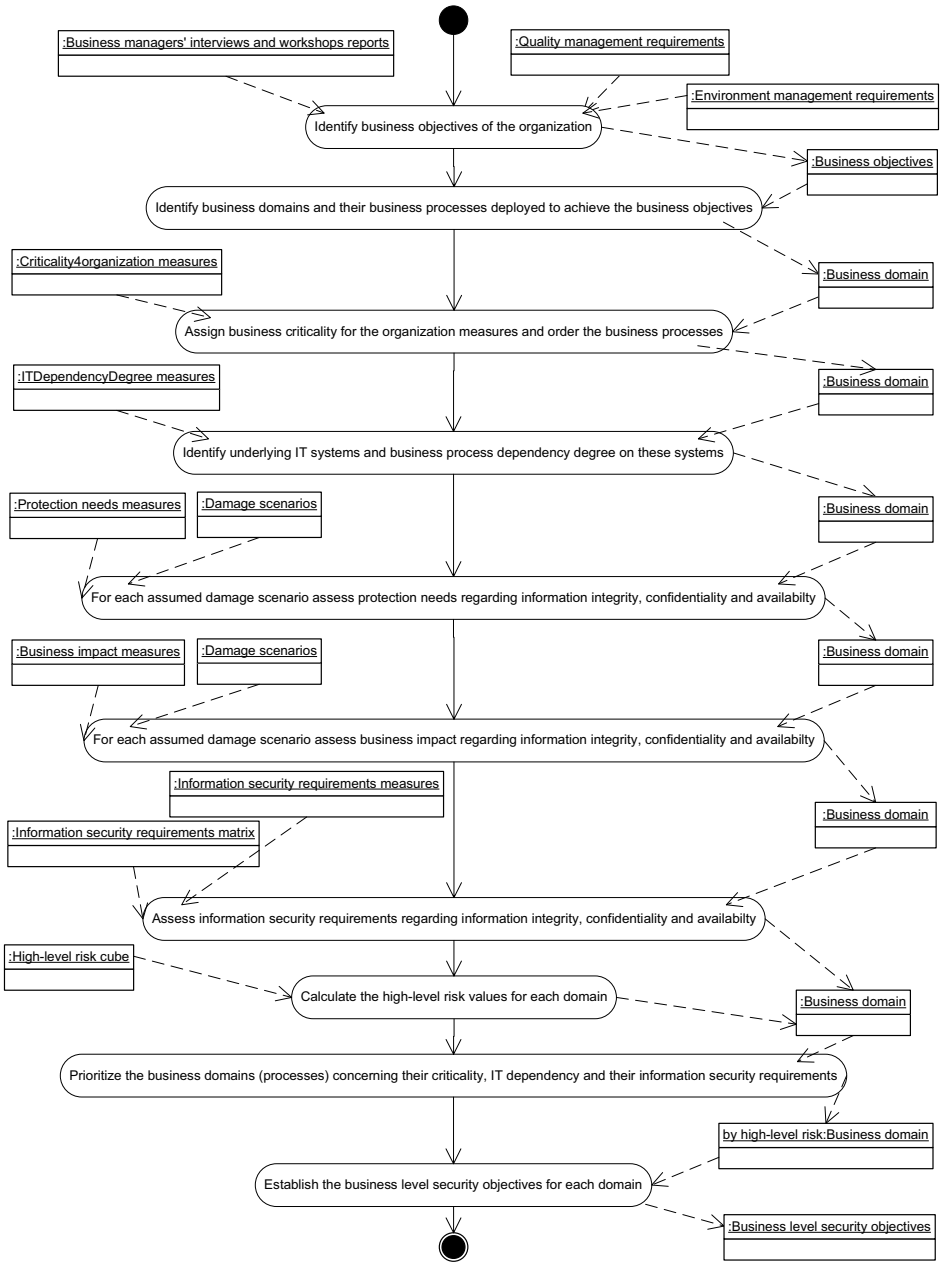


Figure 3. The business environment of the ISMS elaboration

- *High* – The process cannot be performed at all without the support from IT systems and applications,
- *Basic* – The process can be performed partially or slowly (at lower efficiency) or by alternative means at a significant cost (manually, engaging extra personnel, using reserved IT systems, etc.),
- *Low* – The process can be performed by alternative means at a low cost.

These two above mentioned measures can classify all processes from the most to the least critical and IT depending. The third factor taken into consideration are *Protection needs* regarding information security attributes, like integrity, availability and confidentiality. Some processes require high confidentiality but lower availability, others higher integrity with moderate confidentiality and availability, etc. These needs depend only on the character of the operation included in the process.

For *Protection needs* depending on the process essence, let us assume simple measures, the same for each security attributes, as:

- *High* – The process operations require strong information integrity (confidentiality, availability) and cannot be deployed without ensuring this mechanism,
- *Basic* – The process operations require information integrity (confidentiality, availability) at a moderate level, and these needs can be satisfied by alternative ways and means, or later,
- *Low* – The process operations do not require information integrity (confidentiality, availability), these needs are very low, or will arise in the future.

The next step is the prediction of the consequences of losing each of these attributes for the process and the organization. The *Business impact* caused by the loss of these security attributes is performed by “*what-if*” analysis regarding a set of damage scenarios, such as the above mentioned.

Also for the *Business impact* simple measure (the same for each security attributes) the following is proposed [14]:

- *High* – The impact of any loss or damage (caused by the loss of integrity, confidentiality, availability) can bring catastrophic proportions which could threaten the very survival of the organization,
- *Basic* – The impact of any loss or damage may be considerable,
- *Low* – The impact of any loss or damage is limited.

These two factors can be combined together using evaluation matrix (Table 1), but separately for integrity, confidentiality and availability, issuing Information security requirements for each of the security attributes. Please note that Information security requirements use a 5-level measure. Its meaning is implied by the Protection needs and Business impact measures.

Table 1. The matrix of predefined values for information security requirements assessment regarding integrity, confidentiality, availability – separately).

Protection needs	Business impact		
	Low	Basic	High
Low	Low	Basic	High
Basic	Basic	High	Very high
High	High	Very high	Extremely high

All of these factors, expressed in assumed predefined measures, allow to order the business processes with respect to their criticality for organization, IT dependency degree with its information security requirements and their subordinated measures.

These three factors allow to create a global high-level risk measure for business domains of the organization. This three dimensional approach can be expressed as “risk cube” by analogy with the risk matrices. On this basis, filters and risk views can be created allowing the users to trace different risk factors. The number of levels for measures can be user-defined, to better match the organization’s needs.

Example 1: Definition of a global high-level risk measure.

Let us assume the following measures:

- *Criticality for the organization – of the range 1-4,*
- *IT dependency – of the range 1-3,*
- *Information security require (cumulated for all attributes) – of the range 1-5,*

The high-level risk can be defined as the product of these three factors, allowing to measure the high-level risk within the range: 1-60. This measure can be expressed in percentage points as well, and when it would be possible, also in monetary values, as a quantitative measure.

The information security requirements can be considered separately for every attribute, allowing to trace the risk within the integrity/confidentiality/availability cross-sections, or it can be considered globally, using a cumulated measure for all attributes, as it was shown in the Example 1.

Different aspects of the above mentioned risk factors and measures can be used for the following tasks:

- to set the business level security objectives, expressing what should be done to meet the security needs, for all business domains,
- to choose adequate risk management options for the organization or individual business domains,
- to define risk acceptance criteria for the organization,
- to be input data for detailed risk analysis for domains when appropriate,
- to elaborate Plan stage elements.

This approach is a type of a high-level risk analysis compliant with the [13], refined by [14] and used there for partitioning all IT systems of the organization into domains which can be encompassed by the base control, or safeguarded on the basis of risk analysis results.

3. Using Business Environment in the ISMS Elaboration

The results of the above presented high-level risk analysis create a general view of the organization, its needs and risk situation. These preliminary analyses are a good starting point to build an ISMS, and the sampled information is the input for the *Plan* stage elaboration.

The *Plan* stage is focused on three main elements:

- the *ISMS Scope* – presenting boundaries and context of the risk management (Figure 4),
- the *ISMS Policy* – expressing general directions to achieve and maintain information security within the organization (Figure 4),
- the risk management elements – detailed risk analysis and management (Figure 5).

The elaboration of these elements will not be discussed there, instead we will focus on the main dependencies between *Plan* and *Business environment* elements.

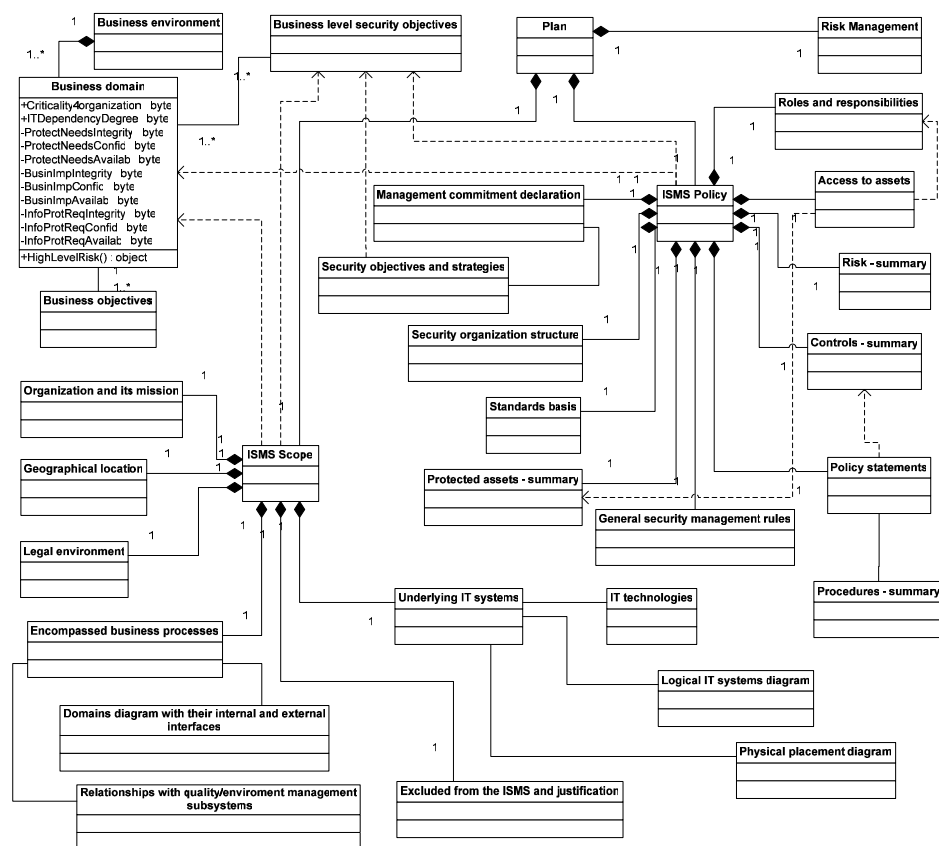


Figure 4. The *Plan* stage elements – the *ISMS scope* and the *ISMS policy*

There is other benefit from previously performed high level risk analysis. The business domains have preliminary identified and evaluated assets of different types [15], threats and needs. It is a good starting point for detailed risk analysis.

4. Conclusions

The paper is focused on the part of the project of the ISMS implementation, dealing with the identification of the business requirements for the ISMS, basing on the high-level risk analysis results.

The concept presented there was used for the development of the prototype of a computer-aided tool [16], [17] supporting information security managers in building and maintaining information security within the organization on the basis of [3] and related standards. The Figure 6 presents a general layout of the high-level risk analyzer, integrated with the ISMS implementation, as the example. For every business domain separate low-level risk analyses can be performed (a smaller window – not considered there).

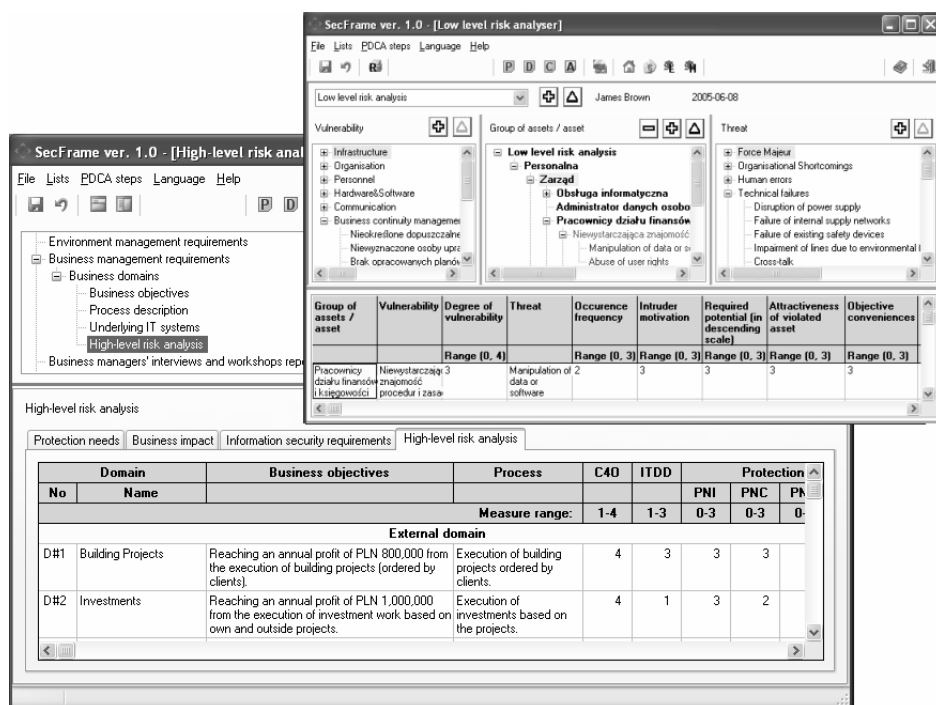


Figure 6. Information security management supporting tool [17]

The current version of the prototype has implemented all main PDCA elements at the basic level, and some details and features are still under development. The first customers' experiences show that the tool is useful during security management – all activities are coordinated, records are supervised, the integrated tools and documents are available quickly. The case studies and the first developers' experiences show that:

- taking advantage of the UML approach is fully possible for the ISMS implementation, as in many other areas of the UML deployment,
- using the UML advantages allows to specify the entire ISMS and its processes in a modular way – methods, measures, tools, document patterns are changeable,
- this flexibility allows tailoring the ISMS according to the size and specific needs of the organization.

In the same way the following are specified: business and its managing processes, security and its managing processes, IT systems and their managing processes and other assets – a unified information security management framework, well positioned within the business environment, basing on the UML approach can be created.

There are two aspects of using the UML approach: firstly, the modeling of security management processes, and secondly, the modeling of software that supports these processes. It can be assumed that the UML approach should be promising for the information security management as well, although this research area does not seem to be well investigated.

Bibliography

- [1] Booch G., Rumbaugh J., Jacobson I.: UML – Przewodnik użytkownika, Wyd. II, *WNT*, Warszawa 2002, (The Unified Modeling Language – User Guide).
- [2] UML site <http://www.omg.org/uml/>
- [3] BS-7799-2:2002 Information security management systems – Specification with guidance for use, *British Standard Institution*.
- [4] Białas A.: IT security modelling, The 2005 International Conference on Security and Management, The World Congress in Applied Computing Las Vegas, June 20-23, 2005.
- [5] Białas A.: Analiza możliwości programowej implementacji systemu typu ISMS, Materiały konferencyjne „Sieci komputerowe 2005”, Politechnika Śląska (in Polish).
- [6] Jürjens J.: Secure Systems Development with UML, *Springer-Verlag*, 2004.
- [7] Galitzer S.: Introducing Engineered Composition (EC): An Approach for Extending the Common Criteria to Better Support Composing Systems, *WAEPD Proc.*, 2003.
- [8] Common Criteria for IT Security Evaluation, Part 1-3, *ISO/IEC 15408*.
- [9] Białas A.: IT security development – computer-aided tool supporting design and evaluation, In: Kowalik J., Górski J., Sachenko A. (editors): *Cyberspace Security and Defense: Research Issues*, NATO Science Series II, vol. 196, *Springer* 2005.
- [10] Lavatelli C.: EDEN: A formal framework for high level security CC evaluations, e-Smart' 2004, Sophia Antipolis 2004.
- [11] Kadam Avinash: Implementation Methodology for Information Security Management System, v.1.4b, *SANS Institute* 2003.
- [12] Białas A.: Bezpieczeństwo informacji i usług w nowoczesnej instytucji, *Wydawnictwa Naukowo-Techniczne*, 2005. (Information security within modern organizations; monograph will be published in 2005).
- [13] ISO/IEC TR 13335-3:1998, Information technology – Guidelines for the management of IT Security, Part3: Techniques for the management of IT Security.
- [14] IT Grundschutz Handbuch, BSI – Bonn: <http://www.bsi.de>
- [15] Białas A.: The Assets Inventory for the Information and Communication Technologies Security Management, *Archiwum Informatyki Teoretycznej i Stosowanej*, Polska Akademia Nauk, tom 16 (2004), zeszyt 2, pp. 93-108, 2004.
- [16] Białas A.: Designing and management framework for ICT Security, Joint Research Centre Cyber-security workshop, Gdansk, 9 - 11 September 2004.
- [17] SecFrame: <http://www.iss.pl>

3. Software Process Methodologies

This page intentionally left blank

Preliminary Analysis of the Effects of Pair Programming and Test-Driven Development on the External Code Quality

Lech MADEYSKI

*Wroclaw University of Technology, Institute of Applied Informatics,
Wyb. Wyspianskiego 27, 50-370 Wroclaw, Poland
e-mail: Lech.Madeyski@pwr.wroc.pl,
<http://madeyski.e-informatyka.pl>*

Abstract. Test-driven development (TDD) and pair programming (PP) are the key practices of eXtreme Programming methodology that have caught the attention of software engineers and researchers worldwide. One of the aims of the large experiment performed at Wroclaw University of Technology was to investigate the difference between test-driven development and the traditional, test-last development as well as pair programming and solo programming with respect to the external code quality. It appeared that the external code quality was lower when test-driven development was used instead of the classic, test-last software development approach in case of solo programmers ($p = 0.028$) and pairs ($p = 0.013$). There was no difference in the external code quality when pair programming was used instead of solo programming.

Introduction

Test-driven development (TDD) [1] and pair programming (PP) [15] have gained a lot of attention recently as the key software development practices of eXtreme Programming (XP) methodology [2].

Researchers and practitioners have reported numerous, often anecdotal and favourable studies of XP practices and XP methodology. Only some of them concern the external code quality (measured by the number of functional, black-box test cases passed) [3], [4] or reliability of programs (the fraction of the number of passed tests divided by the number of all tests) [8], [9], [7]. Systematic review of empirical studies concerning, for example, PP or TDD productivity is out of the scope of this paper but some of these studies are also included in Table 1.

Key findings from empirical studies:

- FP1 Prediction that it takes less time on average to solve the problem by pairs than by individuals working alone was not statistically supported, however the average time for completion was more than 12 minutes (41%) longer for individuals than for pairs [11].
- FP2 Pairs completed their assignments 40–50% faster than individuals [13], [14].
- FP3 Almost no difference in development time between XP-like pairs and solo programmers [10]. PP appeared less efficient than it was reported in [11], [13].

- FP4 A pair of developers does not produce more reliable code than a single developer whose code was reviewed. Although pairs of developers tend to cost more than single developers equipped with reviews, the increased cost is too small to be seen in practice [8], [9].
- FT1 TDD does not accelerate the implementation (working time in minutes) and the resulting programs are not more reliable, but TDD seems to support better program understanding [7].
- FT2 TDD developers produced higher quality code, which passed 18% more functional black box test cases. However, TDD developer pairs took 16% more time for development [3], [4].
- FT3 The productivity (LOC per person-month) of the team was not impacted by the additional focus on producing automated test cases [16] or the impact was minimal [6].

Table 1. Pair programming and test-driven development literature review

study	environment	subjects	findings
Nosek [11]	Industrial	15 (5Pairs/5Solo)	FP1
Williams [13] [14]	Academic	41 (14Pairs/13Solo)	FP2
Nawrocki [10]	Academic	21 (5Pairs/5+6Solo)	FP3
Müller [8] [9]	Academic	37 (10Pairs/17Solo)	FP4
Müller [7]	Academic	19 (9Classic/10TDD)	FT1
George [3] [4]	Industrial	24 (6Classic/6TDD) in 3 trials	FT2
Williams [16] [6]	Industrial	13 (5Classic/9TDD)	FT3

Results of the existing empirical work on PP and TDD are contradictory. This may be explained by the differences in the context in which the studies were conducted. Therefore the context of our experiment is expressed in details to support the development of cause-effect theories and to enable meta-analysis.

In 2004 a large experiment was conducted at Wroclaw University of Technology to study the impact of TDD and PP practices on different aspects of the software development process. This paper examines the impact of TDD and PP on the external code quality.

1. Experiment Definition

The following definition determines the foundation for the experiment:

- **Object of study.** The object of study is the software development process.
- **Purpose.** The purpose is to evaluate the impact of TDD and PP practices on the software development process.
- **Quality focus.** The quality focus is the external code quality.
- **Perspective.** The perspective is from the researcher's point of view.
- **Context.** The experiment is run using MSc students as subjects and finance-accounting system as an object.

Summary: Analyze *the software development process* for the purpose of *evaluation of the TDD and PP practices impact on the software development process* with respect to *external code quality* from the point of view of *the researcher* in the context of *finance-accounting system development by MSc students*.

2. Experiment Planning

The planning phase of the experiment can be divided into seven steps: context selection, hypotheses formulation, variables selection, selection of subjects, experiment design, instrumentation and validity evaluation.

2.1. Context Selection

The context of the experiment was the Programming in Java (PIJ) course, and hence the experiment was run off-line. Java was the programming language, Eclipse was the IDE (Integrated Development Environment). All subjects had experience at least in C and C++ programming (using object-oriented approach). The course consisted of seven lectures and fifteen laboratory sessions. The course introduced Java programming language using TDD and PP as the key XP practices. The subjects' skills were evaluated during the first seven laboratory sessions. The experiment took place during the last eight laboratory sessions (90 minutes per each). The problem (development of the finance-accounting system) was close to the real problem (not toy size). The requirements specification consisted of 27 user stories. In total 188 students were involved in the experiment. The subjects participating in the study were mainly second and third-year (and a few fourth and fifth-year) computer science master's students at Wroclaw University of Technology. A few people were involved in the experiment planning, operation and analysis.

2.2. Quantifiable Hypotheses Formulation

The crucial aspect of the experiment is to know and formally state what we intend to evaluate in the experiment. This leads us to the formulation of the following quantifiable hypothesis to be tested:

- $H_0_{CS/TS/CP/TP}$ — There is no difference in the external code quality between the software development teams using any combination of classic (test last) / TDD (test first) development and solo / pair programming approach (CS, TS, CP and TP are used to denote approaches).
- $H_A_{CS/TS/CP/TP}$ — There is a difference in the external code quality between the software development teams using any combination of classic (test last) / TDD (test first) development and solo / pair programming approach (CS, TS, CP and TP).
- $H_0_{CS/TS}$ — There is no difference in the external code quality between solo programmers using classic and TDD testing approach (CS, TS).
- $H_A_{CS/TS}$ — There is a difference in the external code quality between solo programmers using classic and TDD testing approach (CS, TS).

- $H_{0\ CP/TP}$ — There is no difference in the external code quality between pairs using classic and TDD testing approach (CP, TP).
- $H_{A\ CP/TP}$ — There is a difference in the external code quality between pairs using classic and TDD testing approach (CP, TP).
- $H_{0\ CS/CP}$ — There is no difference in the external code quality between solo programmers and pairs when classic testing approach is used (CS, CP).
- $H_{A\ CS/CP}$ — There is a difference in the external code quality between solo programmers and pairs when classic testing approach is used (CS, CP).
- $H_{0\ TS/TP}$ — There is no difference in the external code quality between solo and pairs when TDD testing approach is used (TS, TP).
- $H_{A\ TS/TP}$ — There is a difference in the external code quality between solo and pairs when TDD testing approach is used (TS, TP).

If we reject the null hypothesis $H_{0\ CS/TS/CP/TP}$ we can try to investigate more specific hypotheses.

NATP (Number of Acceptance Tests Passed) was used as a measure of external code quality. The same measure was used by George and Williams [3], [4]. In contrast to some productivity measures, e.g. source lines of code (SLOC) per person-month, NATP takes into account functionality and quality of software development products. SLOC per unit of effort tend to emphasize longer rather than efficient or high-quality programs. Refactoring effort may even results in negative productivity measured by SLOC.

2.3. Variables Selection

The independent variable is the software development method used. The experiment groups used CS, TS, CP or TP approach.

The dependent (or response) variable is characteristic of the software products on which the factors under examination are expected to have an effect. In our case the dependent variable is NATP.

2.4. Selection of Subjects

The subjects are chosen based on convenience — the subjects are students taking the PIJ course. Prior to the experiment, the students filled in a questionnaire. The aim of the questionnaire was to get a picture of the students' background, see Table 2. The ability to generalize from this context is further elaborated when discussing threats to the experiment. The use of the course as an experimental context provides other researchers opportunities to replicate the experiment.

Table 2. The context of the experiment

context factor	ALL	CS	TS	CP	TP
Number of:					
MSc students:	188	28	28	62	70
— 2nd year students	108	13	16	40	39
— 3rd year students	68	12	11	18	27
— 4th year students	10	3	0	3	4
— 5th year students	2	0	1	1	0

Students with not only academic but also industry experience	33	4	6	8	15
Mean value of:					
Programming experience (in years)	3.8	4.1	3.7	3.6	3.9
Java programming experience (in months)	3.9	7.1	2.8	3.4	3.5
Programming experience in another OO language than Java (in months)	20.5	21.8	20.9	19.2	21.1

2.5. Design of the Experiment

The design is one factor (the software development method) with four treatments (alternatives):

- Solo programming using classic testing approach — tests after implementation (CS).
- Solo programming using test-driven development (TS).
- Pair programming using classic testing approach — tests after implementation (CP).
- Pair programming using test-driven development (TP).

The students were divided into groups based on their skills (measured by graders on an ordinal scale) and then randomized within TDD or classic testing approach groups. However the assignment to pair programming teams took into account the people preferences (as it seemed to be more natural and close to the real world practice). Students who did not complete the experiment were removed from the analysis. The design resulted in an unbalanced design, with 28 solo programmers and 31 pairs using classic testing approach, 28 solo programmers and 35 pairs using TDD practice.

2.6. Instrumentation

The instrumentation of the experiment consisted of requirements specification (user stories), pre-test and post-test questionnaires, Eclipse project framework, detailed description of software development approaches (CS, TS, CP, TP), duties of subjects, instructions how to use the experiment infrastructure (e.g. CVS Version Management System) and examples (e.g. sample applications developed using TDD approach).

2.7. Validity Evaluation

The fundamental question concerning results from an experiment is how valid the results are. When conducting an experiment, there is always a set of threats to the validity of the results. Shadish, Cook and Campbell [12] defined four types of threats: *statistical conclusion*, *internal*, *construct* and *external validity*.

Threats to the *statistical conclusion* validity are concerned with issues that effect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment. Threats to the *statistical conclusion* validity are considered to be under control. Robust statistical techniques, tools (e.g. Statistica) and large sample sizes to increase statistical power are used. Measures and treatment implementation

are considered reliable. However the risk in the treatment implementation is that the experiment was spread across laboratory sessions. To avoid the risk access to the CVS repository was restricted to the specific laboratory sessions (access hours and IP addresses). The validity of the experiment is highly dependent on the reliability of the measures. The measure used in the experiment is considered reliable because it can be repeated with the same outcome. Heterogeneity of the subjects is blocked, based on their grades.

Threats to *internal* validity are influences that can affect the independent variable with respect to causality, without the researcher's knowledge. Concerning the *internal* validity, the risk of rivalry between groups must be considered. The group using the traditional method may do their very best to show that the old method is competitive. On the other hand subjects receiving less desirable treatments may not perform as well as they generally does. However, the subjects were informed that the goal of the experiment was to measure different approaches to software development not the subjects skills. Possible diffusion or imitation of treatments were under control of the graders.

Construct validity concerns generalizing the results of the experiment to the concepts behind the experiment. Threats to the *construct* validity are not considered very harmful. Inadequate explication of constructs does not seem to be a threat as the constructs were defined, before they were translated into measures or treatments – it was clearly stated what having higher external code quality means. The mono-operation bias is a threat as the experiment was conducted on a single software development project, however the size of the project was not toy size. Using a single type of measures is a mono-method bias threat, however the measure used in the experiment was rather objective.

Threats to *external* validity are conditions that limit our ability to generalize the results of our experiment to industrial practice. The largest threat is that students (who had short experience in PP and TDD) were used as subjects. Especially TDD possesses a fairly steep learning curve that must be surmounted before the benefits begin to accrue. However, study by Höst [5] suggest that students may provide an adequate model of the professional population. Furthermore, some of the subjects had also industry experience, see Table 2. In summary, the threats are not considered large in this experiment.

3. Experiment Operation

The experiment was run at Wroclaw University of Technology in 2004 during eight laboratory sessions. The data was primarily collected by automated experiment infrastructure. Additionally the subjects filled in pre-test and post-test questionnaires, primarily to evaluate their experience and preferences. The package for the experiment was prepared in advance and is described in Section 2.6.

4. Analysis of the Experiment

The experiment data are analyzed with descriptive analysis and statistical tests.

4.1. Descriptive Statistics

A good way to display the results of the experiment is by using a box and whisker diagram or box plot shown in Figure 1. The box represents the range within which 50% of the data fall. The point within the box is the median. The 'I' shape shows us the limits within which all of the data fall. The first impression is that classic approaches performed better than TDD approaches.

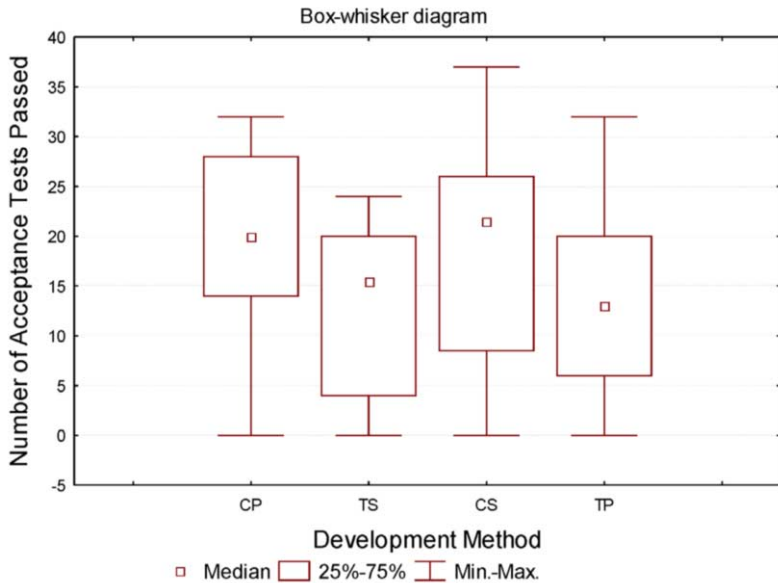


Figure 1. Box-whisker plot for the number of acceptance tests passed in different development methods

4.2. Hypotheses Testing

Experimental data are analysed using models that relate the dependent variable and factor under consideration. The use of these models involves making assumptions about the data that need to be validated. Therefore we run some exploratory analysis on the collected data to check whether they follow the assumptions of the parametric tests:

- Interval or ratio scale – the collected data must be measured at an interval or ratio level (since parametric tests work on the arithmetic mean).
- Homogeneity of variance – roughly the same variances between groups or treatments (if we use different subjects).
- Normal distribution – the collected data come from a population that has a normal distribution.

The first assumption is met. The second assumption of homogeneity of variance is tested using Levene's test, see Table 3. The Levene test is non-significant ($p > 0.05$) so we accept the null hypothesis that the difference between the variances is roughly zero – the variances are more or less equal.

Table 3. Test of Homogeneity of Variances

Levene Statistic	Significance
1.199	0.313

Having checked the two assumptions we have to test the third one — normality assumption using the Kolmogorov-Smirnov test as well as the Shapiro-Wilk test.

We find that the data are not normally distributed in case of CS approach (according to the Kolmogorov-Smirnov statistic), TS and CP (according to the Shapiro-Wilk statistic), see Table 4. This finding alerts us to the fact that a nonparametric test should be used.

Table 4. Tests of Normality

Approach	Kolmogorov-Smirnov ¹			Shapiro-Wilk		
	Statistic	df ²	Significance	Statistic	df ²	Significance
CS	0.182	28	0.018	0.931	28	0.066
TS	0.157	28	0.075	0.893	28	0.008
CP	0.116	31	0.200 ³	0.912	31	0.014
TP	0.111	35	0.200 ³	0.960	35	0.222

The hypothesis regarding the difference in external code quality between the software development teams using CS, TS, CP and TP approach is evaluated using the Kruskal-Wallis one way analysis of variance by ranks. The Kruskal-Wallis test is a non-parametric alternative to the parametric ANOVA and can always be used instead of the ANOVA if it is not sure that the assumptions of ANOVA are met. The Kruskal-Wallis test is used for testing differences between the four experimental groups (CS, TS, CP, TP) when different subjects are used in each group.

The Kruskal-Wallis test analyses the ranked data. Table 5 shows a summary of these ranked data and tells us the mean rank in each treatment. The test statistic is a function of these ranks.

Table 5. Ranks

Treatment	N	Mean Rank
CS	28	69.46
TS	28	50.79
CP	31	74.58
TP	35	52.11
Total	122	

¹ Lilliefors Significance Correction
² Degrees of freedom
³ This is a lower bound of the true significance

Table 6 shows this test statistic and its associated degrees of freedom (in this case we had 4 groups so $4 - 1$ degrees of freedom), and the significance.

Table 6. Kruskal-Wallis Test Statistics (grouping variable: Approach (CS,TS,CP,TP))

	NATP
Chi-Square	10.714
df	3
Asymp. Significance	0.013

We can conclude that the software development approach used by the subjects significantly affected the external code quality (measured by NATP). This test tells us only that a difference exists, however it does not tell us exactly where the difference lies.

One way to see which groups differ is to look at the box plot diagram of the groups (see Figure 1). The first thing to note is that classic approaches (CS, CP) achieved better results (higher numbers of acceptance tests passed) than TDD approaches (TS, TP). However, this conclusion is subjective. We need to perform planned comparisons (contrasts) or multiple independent comparisons (Mann-Whitney tests) for specific hypotheses from Section 2.2. It is justified as we identified the comparisons (specific hypotheses) as valid at the design stage of our investigation. The planned comparisons, instead of comparing everything with everything else, have the advantage that we can conduct fewer tests, and therefore, we don't have to be quite strict to control the type I error rate (the probability that a true null hypothesis is incorrectly rejected).

Tables 7, 8, 9 and 10 show the test statistics of Mann-Whitney tests on the four focused comparisons. When CP and TP approaches are compared the observed significance value is 0.013, see Table 8. When CS and TS approaches are compared the observed significance value is 0.028, see Table 7. The planned contrasts also suggest that using TDD instead of classic testing approach decreases the external code quality in case of pairs as well as solo programmers ($p < 0.05$).

Table 7. Mann-Whitney Test Statistics (CS vs. TS)

	NATP
Mann-Whitney U	258.000
Wilcoxon W	664.000
Z	-2.198
Asymp. Significance (2-tailed)	0.028

Table 8. Mann-Whitney Test Statistics (CP vs. TP)

	NATP
Mann-Whitney U	348.500
Wilcoxon W	978.500
Z	-2.495
Asymp. Significance (2-tailed)	0.013

Table 9. Mann-Whitney Test Statistics (CS vs. CP)

	NATP
Mann-Whitney U	393.500
Wilcoxon W	799.500
Z	-0.615
Asymp. Significance (2-tailed)	0.538

Table 10. Mann-Whitney Test Statistics (TS vs. TP)

	NATP
Mann-Whitney U	485.000
Wilcoxon W	1115.000
Z	-0.069
Asymp. Significance (2-tailed)	0.945

5. Summary and Conclusions

The external code quality (measured by a number of acceptance tests passed) was significantly affected by the software development approach (the Kruskal-Wallis test statistics: $H(3) = 10.71$, $p < 0.05$ where H is the test statistic function with 3 degrees of freedom and p is the significance). This means that there is a difference in the external code quality between the software development teams using CS, TS, CP and TP approach. Mann-Whitney tests were used to follow-up this finding. It appeared that the external code quality was lower when TDD was used instead of the classic, test-last software development approach in case of solo programmers (Mann-Whitney CS vs. TS test significance value $p = 0.028$) and pairs (Mann-Whitney CP vs. TP test significance value $p = 0.013$). There was no difference in the external code quality when pair programming was used instead of solo programming (Mann-Whitney CS vs. CP test significance value $p = 0.538$, Mann-Whitney TS vs. TP test significance value $p = 0.945$). The validity of the results must be considered within the context of the limitations discussed in the validity evaluation section.

Future research is needed to evaluate other properties than the external code quality as well as to evaluate PP and TDD in other contexts (e.g. in industry).

Acknowledgments

The author would like to thank the students for participating in the investigation, the graders for their help and the members of the e-Informatyka team (Wojciech Gdela, Tomasz Poradowski, Jacek Owocki, Grzegorz Makosa, Mariusz Sadal and Michał Stochmiałek) for support and preparation of the infrastructure for the experiment to automate measurements which appeared extremely helpful.

The author also wants to thank prof. Zbigniew Huzar and dr Małgorzata Bogdan for helpful suggestions.

References

- [1] Beck, K.: Test Driven Development: By Example. *Addison-Wesley* (2002)
- [2] Beck, K.: Extreme Programming Explained: Embrace Change. 2nd edn. *Addison-Wesley* (2004)
- [3] George, B., Williams, L.A.: An initial investigation of test driven development in industry. In: *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC)*, ACM (2003) 1135—1139
- [4] George, B., Williams, L.A.: A structured experiment of test-driven development. *Information & Software Technology* **46** (2004) 337–342
- [5] Höst, M., Regnell, B., Wohlin, C.: Using students as subjects — a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering* **5** (2000) 201–214
- [6] Maximilien, E.M., Williams, L.A.: Assessing Test-Driven Development at IBM. In: *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, IEEE Computer Society (2003) 564—569
- [7] Müller, M.M., Hagner, O.: Experiment about test-first programming. *IEE Proceedings – Software* **149** (2002) 131–136
- [8] Müller, M.M.: Are reviews an alternative to pair programming? In: *Proceedings of the Conference on Empirical Assessment In Software Engineering (EASE)*. (2003)
- [9] Müller, M.M.: Are reviews an alternative to pair programming? *Empirical Software Engineering* **9** (2004) 335–351
- [10] Nawrocki, J.R., Wojciechowski, A.: Experimental evaluation of pair programming. In: *Proceedings of the European Software Control and Metrics Conference (ESCOM)* (2001) 269–276
- [11] Nosek, J.T.: The case for collaborative programming. *Communications of the ACM* **41** (1998) 105–108
- [12] Shadish, W.R., Cook, T.D., Campbell, D.T.: Experimental and Quasi-Experimental Designs for Generalized Causal Inference. *Houghton Mifflin* (2002)
- [13] Williams, L., Kessler, R.R., Cunningham, W., Jeffries, R.: Strengthening the case for pair programming. *IEEE Software* **17** (2000) 19–25
- [14] Williams, L.: The Collaborative Software Process. *PhD thesis*, University of Utah (2000)
- [15] Williams, L., Kessler, R.: Pair Programming Illuminated. *Addison-Wesley* (2002)
- [16] Williams, L.A., Maximilien, E.M., Vouk, M.: Test-driven development as a defect-reduction practice. In: *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE)* (2003), IEEE Computer Society (2003) 34–48

Codespector – a Tool for Increasing Source Code Quality

Mariusz JADACH and Bogumiła HNATKOWSKA

Wrocław University of Technology,

Institute of Applied Informatics,

Wrocław, Poland

e-mail: bogumila.hnatkowska@pwr.wroc.pl

Abstract. Software quality is strictly connected with the source code quality. Coding standards may be very helpful in order to assure the high quality of the code, especially that they are supported by many application tools. The paper presents Codespector – the tool for checking the Java source files against a given coding standard. The rules of the standard are written using the special CRL language, elaborated by one of co-authors.

Introduction

A quality is one of the most important and desired properties of the software. Software Quality Assurance (SQA), and Verification and Validation (V&V) are the major processes bear directly on the quality of the software product [1]. The SQA and V&V use static and dynamic techniques for quality providing. Static techniques involve examination of different kind of documentation, one of them is the source code. These techniques often are supported by automated tools.

The paper presents an idea of the new tool – called Codespector – that aims in increasing quality of the source code in Java language. The idea proposes a new way of checking source code against some coding standard. The tool was designed and implemented as the main part of Master Thesis of one of co-authors.

Section 1 describes the role of programming style and coding standards.

There are many tools for validating source files, that check the source code, whether it conforms the standards or not. Some of them are briefly presented in Section 2.

Section 3 presents the idea of Codespector tool. The important part of the project was designing a CRL language, which is the integral part of the tool. A simple case study of using the tool is also presented.

Some concluding remarks are gathered in Section 4.

1. Coding Conventions

A programming style (coding standards, coding conventions) is defined as a set of “conventions for writing source code in certain programming language” [2]. These conventions have a form of very practical (usually) rules, which tell programmers how to code some elements of the program or what programming techniques to use. For

example, typical code convention is a requirement that private components of the class (only in object-oriented languages) are always located at the beginning of this class declaration, then protected components, and public components at the end. Of course, there is no common, obligatory coding standard for every language. Everyone may define his own set of coding rules, and these rules may be different or inconsistent with conventions from an other standard. In spite of this, there are many useful, existing coding standards including valuable rules, which were checked in practice. Example of such a popular coding style is the document “Java Coding Conventions” [3], available free from the Internet.

There are many important reasons for using programming styles. First of all, it is a very good and easy way to improve the code quality and finally, the software quality. For example, papers [3] and [4] indicate that the programming styles increase maintainability and readability of the code, that is crucial fact in context of team work. However, to gain success it is necessary to use the defined standard consistently by every person in a team [3].

2. Tools for Examining Coding Standards

Today, the coding conventions are very well known and widely used. Many different application tools have been created in order to help the programmers to apply the standards in practice. A few of such tools are described briefly in this section.

2.1. Checkstyle

Checkstyle is a quite well known tool for checking applied coding standard in Java language source code. The tool is developed in Java as an open source project. The last version of the tool (3.5) is free available from the project site in the Internet [5]. Checkstyle is a command line tool, but many different plug-ins were created to use the tool in popular Java IDEs [6]. For example, Checklipse and Eclipse Checkstyle are plug-ins for the Eclipse environment.

The Checkstyle tool validates Java source files according to given configuration. The configuration is a XML file, which includes the hierarchy of *modules*. The root module is “Checker”. Its submodule “TreeWalker” is especially important, because it contains the modules of the particular *checks*. These checks are applied to the corresponding fragments of the abstract syntax tree of the analyzed source file. Many of such checks are present within the tool. Each check may be customized by setting the values of its properties. The built-in checks are well described in [6].

Defining own checks is also possible in the Checkstyle tool. A Java class, which represents a new check must be provided by the user. It extends the `Check` class. The visitor design pattern is the basis of analyzing source files, so the method `visitToken` must be implemented. This method is invoked every time when the corresponding code fragment (token) occurs. In order to use the own check, it must be integrated with the configuration file by passing the name of the created class as the new module. More information about extending the tool and writing own checks is available in the documentation [6].

2.2. PMD Project

PMD is also developed as an open source project. It serves for validation of Java source. The base distribution of the tool supports the command line usage, but many plug-ins or extensions were created for popular Java IDEs (Eclipse, NetBeans, JBuilder, etc.) [7].

The checks of the source file are described in XML files, which are called *rulesets*. The built-in rules of the code are categorized and divided into a number of rulesets (e.g. basic, naming, braces, clone, etc.).

User's own rulesets may be also created. These files may include references to the existing rules from other rulesets (built-in or user's). Some properties of each rule (like message or priority) may be customized.

The possibility of creating own rules is also supported in the PMD tool. There are two ways of doing this. First of all, the user may write and deliver Java class (extending *AbstractRule*). The method *visit* is invoked for the corresponding syntax elements of the code (exactly for the nodes of the abstract syntax tree). The method's body is responsible for checking these elements according to the rule conditions. The visitor design pattern is also the basis of analysis by the PMD tool. The latter way of defining own rules is creating the XPath query expressions. Finally, the user's rules must be integrated into some ruleset by writing new XML elements in the ruleset file.

2.3. ESQMS

A very interesting tool is described in the paper [8]. The Electric Power Research Institute (EPRI) in cooperation with Reasoning Systems built "a prototype toolset called EPRI Software Quality Measurement System (ESQMS) for measuring compliance of software with EPRI's coding standards" [8].

Refine/C is a special reengineering workbench for C programming language, developed by Reasoning Systems [8]. This workbench is based on the other environment called Software Refinery, built by Reasoning Systems too. Refine/C is the base of the ESQMS tool. The source code in C language is represented as the abstract syntax tree (AST) within the workbench. The rules, which correspond to the particular coding standards are defined and written using a special, very high level language called Refine [8].

The Refine language is complex and powerful. Mathematical operations, AST tree operations, transformations of the code and pattern-matching of AST are supplied [8]. Functions may also be defined. The language and the workbench was especially designed to build different tools for analyzing and transforming the C code. The rules, described in Refine language are similar to functions. A node of the AST tree (currently analyzed) is passed as their argument. The rule contains a set of preconditions and a set of postconditions. Each of the postconditions must be true in case of all the preconditions are true. Typically, the preconditions describe the requirements of the coding standard (e.g. the node must be an equality operator), and the postconditions describe the results of checking them (e.g. adding the tuple <node, message> into the violations set). Some automatic transformations of the code (through transformation of AST) are also possible. The postconditions are used to write the correct form of the analyzing code fragment. More detailed informations about the ESQMS and the Refine language may be found in the mentioned paper [8].

3. The Codespector Tool

3.1. The Idea of the Tool

The Codespector tool presented in the paper is an application for analyzing a Java source file. The analysis is done in order to check conforming to some coding standards. These standards are described as a set of rules (*ruleset*) which is stored in a text file, called *rules file*. Rules are written in a special language, called Code Rules Language (CRL).

The collection of the built-in coding rules is usually available in the most of the existing tools for checking conforming to coding standards. These rules may be used to build an own programming style. Creating own rules is also possible in many of the tools [4], [7], [8]. It is a very useful and powerful function. That is why the basic aim of the proposed tool is to enable the user to express his or her coding rules in easy, consistent and comfortable way.

Creating own rules in many of the existing tools may be quite sophisticated or boring task. Providing a special, external class definition with some implemented methods is the common way of defining the user's rule. The knowledge about the API of the analyzing source code and about some programming language is of course required. What is worse, some modifications of the textual configuration file (usually special XML file) are also necessary to add such a custom rule (the class) to the set of standard's rules. The syntax and semantics of the file elements must be known. It may be noticed, that this approach results in "mixing" of the different languages. Besides, setting some classpaths or putting the external classes into a special folder may be required for proper linking the user's rules.

The solution may be the CRL language. The whole definition of the rule is written and saved in the rules file. One ruleset is contained within one file. The idea of the CRL language is similar to the Refine language, which was described in previous chapter (ESQMS tool). But the CRL was intent to have more clear and more simple syntax.

The rules files in the CRL language are used by the tool as the entry data. The analyzed Java source code is the second entry data for the tool. Scanning and parsing the source file is the first phase of the source code analysis.

The analyzed representation of the code must be completely compatible with the CRL language. The code structure is stated by the *tree of parts* and the *list of tokens*, which are connected with each other. The list of tokens (equivalent to lexemes) is the result of scanning (lexical analysis) the source code. The *code part* is a higher level fragment of the code. It is a syntax unit produced by the parser (syntax analysis).

Every item of the code structure (token or part) is visited and analyzed during the analysis (exactly once). Each of them is potentially a *triggering event*, which may trigger checking the associated rules. The active ruleset is searched in order to find such a corresponding rule. Exactly one triggering event is defined for every rule. The triggering event is simply the name of the code fragment's type (e.g. keyword, class, method etc.). The visited code fragment – triggering event is the *current context* of the analysis. This is used for checking the fulfilling of the rule requirements. A variety of rules may be defined for one triggering event.

The *validation report* is the main result of the Codespector's analysis. The information about found violations of coding standards is included in the report and may be inspected by the user.

3.2. The CRL Language

CRL is a declarative and specialized language used to describe the ruleset in a clear and easy way. One ruleset is included in one ASCII text file, which may be created or edited by the user. The structure of this file is strictly defined. It is briefly presented below.

The descriptive information about the ruleset is included at the beginning of the file. It is stated by the values of the five properties, which are written as *name=value* pairs. These properties are: *name of the ruleset*, *name of the ruleset's author*, *file version*, *compatibility with the tool version* and *textual description of the file content*.

The particular rules are defined in the rest part of the file. One or more rule definitions may be contained within the file. Each rule definition is constructed by four sections. Each section has a name (CRL keyword(s)) and a body. The body of the section is always included between the pair of the brace brackets.

The first section (*rule section*) begins with the keyword *rule*. Only the name of the rule is contained within the section body.

The second section is the *for section*. It specifies a triggering event for the rule. The optional *entry conditions* of the rule may be included in the section body. The entry conditions define the conditions that must be met for further analysis of the current rule. These conditions may be used as a kind of filter for analyzed code fragments.

The body of *for* section consists of boolean *condition expressions*. The expressions are separated by semicolon within the section. The results of the particular expressions are connected by logical operator *and*, giving the final logical result of the all section conditions. An individual condition expression may be composed using many nested expressions, which are connected by logical operators (*and*, *or*, *not*). The nested expression must be included within the pair of round brackets.

The unnested condition expression has the form of *built-in condition* or *external condition*. The built-in condition is the comparison of two *sides*, which represent some values. All values in the CRL are *objects*. The comparison operators are: *=*, *<*, *<*, *>*, *<=*, *>=*, *in*. The set of correct operators in the current context is determined by the types of values on the two sides of the comparison. Six data types are defined within the CRL. The simple types are: *boolean* (logical), *text* (any quoted strings) and *number* (integral and floating). The values of these types may be directly used (written) in the ruleset file. The complex types in the CRL are: *set* (a set of text or number values only), *token* and *part*. The last two types are used for representing the fragments of the analyzed code and may not be used directly as a written value. The special value *null*, which is relevant to empty object, is defined for all types.

The *function list* is always the left side of the comparison. The current context of the rule analysis is represented by the keyword *this* that is always put at the beginning of the list. The current context is the syntax unit of the code – the triggering event of the rule. Many subsequent functions' invocations may occur following the keyword *this*. They are separated by the dot symbol, which means the reference to object's (value's) function, analogous to object-oriented programming languages. Each of the functions in the function list is invoked for the object – the result of invoking the previous (on the left) function. Of course, the type of the first value in the list (keyword *this*) is *part* or *token*. The result of the last function's invocation in the list is used for comparison as the final value of the whole list. The right side of the comparison may be another function list or some direct written value.

The set of functions is part of the CRL language specification. Each function is invoked for objects of one of CRL type. The functions are identified by their names and must return a value of the CRL type. Optional function's arguments may be also defined. The *part* and *token* types are specific. Their current sets of available functions are additionally dependent on current category of the code fragment (e.g. a value of the type *token* may be included in the category *keyword*, *operator*, etc.). An error is generated on attempt to invoke the function, which does not exist or is not available for the object. Similarly, invoking any function for the *null* object is not allowed too.

There is also possibility to write some user's conditions in Java language as the external classes, which may be linked to rules in the ruleset file. The external condition is used for this purpose. The CRL keyword *externalClass* may be put as the part of the condition expression, associated with the fully-qualified name of the external class. The special Java interface must be implemented by the class in order to call interface's method during conditions evaluation. The result of this method is logical and indicates whether the external condition is fulfilled.

The third section of the rule is the *check section*. The *check conditions* are defined within it. These conditions are used to verify the conformance of the current context (code fragment) to the rule. All of the conditions within the section must be met in order to assure that the analyzed fragment meets the rule requirements. The syntax and the semantics of the check conditions are analogous to entry conditions. At least one condition expression must be contained within the check section.

The last part of the rule is the *action section*. The message about the rule violation (*message*) and the rule severity (*kind*) is included there. The three levels of the severity are defined: *note*, *warning* and *error*. The message and the rule severity are used to generating the validation report. Optionally, the action section may define some *external action* in the form of invoking special method of the provided, external Java class. The external action is executed when the code rule is not fulfilled.

The traditional comments in C style (*/* ...*/*) are supported in the CRL language. They may be put anywhere in the file. The language is case sensitive.

Let us present two simple examples of defining code rules using the CRL language. The content of the example ruleset file is written below.

```
plugin = "1.0";
name = "Test file";
version = "1.0";
author = "Mariusz Jadach";
description = "It is just an example";

rule /* Rule 1 */
{
    name = "Rule 1: using magical numbers";
}
for integer
{
    this.getAncestorLevel("field") = null;
}
check
{
    this.getTokenText in { "0", "1" };
}
action
```

```

{
    message = "You should always declare numbers
              other than 0 or 1 as constant values";
    kind = note;
}

rule /* rule 2 */
{
    name = "Rule 2: block is obligatory";
}
for if
{
}
check
{
    (this.getThenPart.getPartType = "block" and
     (this.getElsePart = null
      or this.getElsePart.getPartType = "block"));
}
action
{
    message = "Instructions within if should be
              always written in a block";
    kind = error;
}

```

Presented Rule 1 checks if the integral values, different than 0 or 1 are written explicite in the code. Using many “magical” numbers in the source code is not good practice (floating-point numbers too) [9]. Modifications of them may be error-prone and troublesome. Every number should be used as constant or variable.

The triggering event of the rule is *integer*. The rule is checked when some integral number is spotted in the source code. The condition in the for section checks whether the analyzed number is a part of the class field declaration (as field’s initial value). In such a case the number may appear in the code. The *getAncestorLevel* function with argument “field” returns the number of levels in the parts tree between the *integer* token and closest *field* part. When the result of the function is *null*, the token does not have an ancestor of the category *field*.

Only one condition expression is included in the check section. This is the unnested expression. The function list on the left side of the comparison is very simple. The invocation of the function *getTokenText* is performed for the current context value. The result of the invocation (of text type) is compared with the right side of the comparison using the operator *in*. The *in* operator returns true if the left side is a member of the set, defined on the right side. The set of two text values is on the right side. The rule is violated when the text of analyzed number (token) is different from 0 or 1.

The second rule verifies whether the statements within *if* instruction are included in the block (compound statement). Putting statements in the block may help in adding new statements without introducing errors [3].

The triggering event of the rule is *if*, because *if* statements in the code must be checked. The *for* section is empty, so each *if* statement is further analyzed in the *check* section.

The condition expression within the check section is nested. The two partial expressions are connected by *and* operator. The first expression checks whether the

statement executing in the normal case is compound statement. The code fragment, corresponding to the statement running in the normal case may be obtained by invoking the CRL function *getThenPart*. The function *getPartType* is used to getting the name of the fragment's category – it must be “block”.

The second expression checks whether the *else* action of the *if* statement is also a compound statement. The analogous method of evaluation is used as in the first condition expression. Existing of the *else* fragment is obviously necessary. That is why the additional checking for the *null* value is present.

3.3. The Project of the Tool

The Codespector tool is designed as plug-in for the Eclipse environment (very popular IDE). The architecture of the Codespector tool is briefly described in this section. The architectural project is mainly determined by the functionalities and the properties of the proposed tool.

The four architectural parts (modules) of the program are identified. They are presented in the Figure 1. The three of them are functional modules: *compilation module*, *structure module* and *validation module*. These modules are also called the *core* of the tool. The *user interface module* is responsible for interaction between the core and its environment. Compilation of the input CRL rules file is the main role of the compilation module. Syntax errors and some semantic errors are detected by the compiler. This compiler is automatically generated by the Grammatica tool (parsers generator). The abstract syntax tree of ruleset file is created during the process. The tree is used to construct the internal representation of the ruleset.

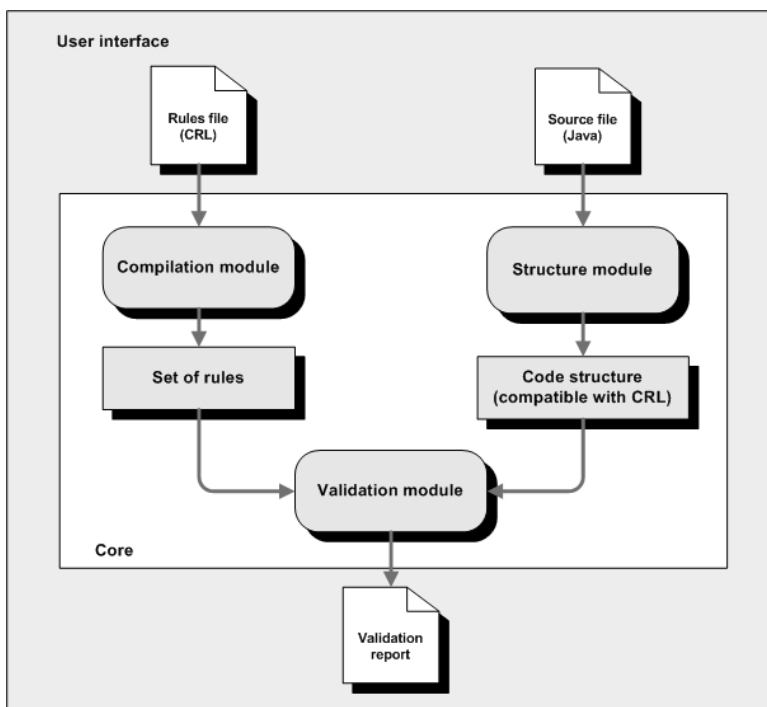


Figure 1. The architecture of the Codespector tool

The structure module is used to transform the input source code to corresponding, internal representation, which is compatible with the CRL language. The Java *tokenizer* (lexical scanner) included in Eclipse environment JDT (Java Development Environment) is used within the module. The JDT classes are also used to get the AST of the analyzed source code. The resulting syntax tree and the list of tokens must be transformed to get the parts' tree and the list of tokens, which are compatible with the CRL. Then they are merged in order to build the one consistent code structure.

The main functionality of the tool is realized by the validation module. The input data of this process are: the internal representation of the compiled CRL ruleset file and the internal representation of the analyzed source code (structure of the code). The visitor design pattern is used to browse the code structure and visit each token and code part. The set of corresponding rules is found for every such a code fragment. These rules are applied to the current analyzed fragment in order to check its conformance to the rules. Any violations of the rules are saved by adding some adequate information to the validation report.

The user interface of the tool is designed to interact with the plug-in user and with the plug-in environment (Eclipse IDE). The current user interface is Eclipse specific, but it may be changed into other user interface (e.g. command line UI). The core of the tool is completely independent from the used particular user interface.

The Codespector tool is already implemented. On the Figure 2 an example validation report is presented. The report lists violations of rules defined in the uploaded rulset file along with the information of their type (i.e. error, warning, note). A user may navigate in an easy way to the code that violates a given rule.

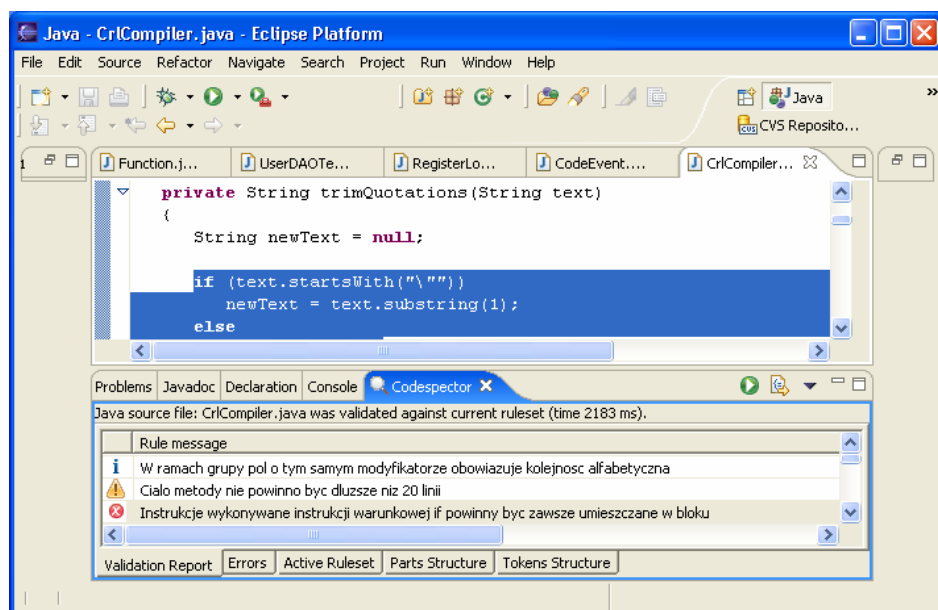


Figure 2. The snapshot of the main window of the Codespector tool

3.4. Verification of the Tool

The functions of the Codespector were verified by manual testing. For that purpose a file with 15 different code rules relating to real recommendations of good style of programming was prepared. This ruleset file was used for analysis of 5, arbitrary chosen java files. All the tested files come from different open-source projects. The files have different lengths. They were:

- jdbcDatabaseMetaData.java – the file from the relational database system HSQLDB (<http://hsqldb.sourceforge.net>)
- CppCodeGenerator.java – the file from the ANTLR project – a parser generator (<http://www.antlr.org>)
- WorkbenchWindow.java – the file from the Eclipse project
- Emitter.java – the file from Jflex tool – a free parser generator (<http://jflex.de>)
- ButtonDemo.java – the file from an explanatory program, presenting the usage of the Swing library (standard distribution of JDK 1.4.1)
- TestClass.java – the file, prepared by the author

The number of all violations, and the number of violations of a given type (i.e. error, warning, note) for particular test files are summarized in Table 1. The Table 2 presents the number of violations of each rule for different test files.

Table 1. The number of all violations and violations of a given type

File name	All violations	Violations of a given type		
		error	warning	note
jdbcDatabaseMetaData.java	80	0	10	670
CppCodeGenerator.java	3483	134	77	3272
WorkbenchWindow.java	1565	62	58	1445
Emitter.java	1459	90	69	1300
ButtonDemo.java	865	7	18	840
TestClass.java	56	2	12	42

How to foresee easily, the file prepared by the author violated every rule of code. The most often violated rule was the rule number 8. This rule concerns applying spaces between parentheses and the content. The rule is controversial and rarely used.

The code rules, presented in details in the previous section are described also in Table 2 as the rule 11 (magic numbers), and the rule 7 (if statement).

4. Summary

The quality of the source code determines the total quality of the final software product. Using programming style is a good practical approach to improve the code quality. The process of checking conformity of the code to the programming style can be automated by different tools.

Table 2. The number of violations of particular rules for different test files

File name	Rule number														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
jdbcDatabaseMetaData.java	1	1	0	1	0	0	0	666	0	0	2	0	7	2	0
CppCodeGenerator.java	3	16	0	12	1	20	134	3157	24	0	66	1	44	5	0
WorkbenchWindow.java	2	17	0	3	3	0	62	1413	28	0	9	3	22	0	3
Emitter.java	0	10	0	17	2	5	90	1248	39	0	18	2	27	1	0
ButtonDemo.java	0	8	0	0	0	0	7	828	11	1	3	0	7	0	0
TestClass.java	3	5	1	2	2	6	2	15	2	1	12	1	2	1	1

The paper presents the Codespector tool, working as the plug-in to the Eclipse IDE for checking Java source files. The flexibility and simplicity of defining user's rules, being the parts of the own code standard, were the basic motivations to design and implement it. That's why, the most important element of the proposed solution is the CRL language. The CLR language enables the user to express many rules in an easy way. This language is the element which differs Codespector from other solutions.

The tool will be further developed. For example, the automatic fixing of the code is planned in the next release. Many improvements and extensions may be done for the CLR language. For example, enhancing the syntax of condition expressions (complex sides of comparison, additional operators, etc.) will be worthwhile.

References

- [1] IEEE Computer Society: Guide to the Software Engineering Body of Knowledge — SWEBOK, Los Alamitos, 2004. Available at <http://swebok.org>
- [2] The Free Dictionary, <http://encyclopedia.thefreedictionary.com>
- [3] Sun Microsystems, Inc., Java Code Conventions. Available at <http://java.sun.com/docs/codeconv>
- [4] Ambler, Scott W.: Writing Robust Java Code, *The AmbySoft Inc.* Coding Standards for Java, AmbySoft Inc., 2000.
- [5] The official Checkstyle project site at SourceForge. Available at <http://sourceforge.net/projects/checkstyle>
- [6] Documentation of the Checkstyle tool. Included as a part of the tool distribution.
- [7] The official PMD project site at SourceForge. Available at <http://pmd.sourceforge.net>
- [8] Wels, Charles H., Brand Russel., Markosian Lawrence.: Customized Tools for Software Quality Assurance and Reengineering, *Second Working Conference on Reverse Engineering*, July 14-16, 1995, Toronto, Canada.
- [9] McConnell Steve: Code Complete, Second Edition, *Microsoft Press*, Redmond, 2004.

Software Complexity Analysis for Multitasking Systems Implemented in the Programming Language C

Krzysztof TROCKI ^{a,b} and Mariusz CZYŻ ^a

^a *Motorola Polska Electronics*

e-mails: {krzysztof.trocki, mariusz.czyz}@motorola.com

^b *Silesian University of Technology, Department of Computer Science,*
e-mail: krzysztof.trocki@polsl.pl

Abstract. The paper investigates the use of two of the most popular software complexity measurement theories, the Halstead's Software Science metrics and McCabe's cyclomatic complexity, to analyze basic characteristics of multitasking systems implemented in the programming language C. Additional extensions of both systems of metrics are proposed to increase the level of obtained information connected with the typical characteristics of multitasking systems.

Introduction

One of the main problems associated with existing theories describing various approaches to the software complexity measurement is that most of them have been developed in order to calculate metrics for sequential software. An analysis of complexity of software employing various techniques of parallel processing becomes much more difficult and also much more desirable as the creation of such type of software is definitely more intellectually challenging.

Since the software complexity assessment has been a subject of numerous studies, the perspective applied in this paper has been limited by the following factors:

- choice of metrics calculated directly on the basis of source code, which allow to use automated software analysis tools,
- choice of theories, which have been widely studied and empirically verified,
- choice of theories, which have become the inspiration for the further research.

Using the above perspective, two main widely studied and utilized theories have been chosen for the further analysis: Halstead's Software Science and McCabe's cyclomatic complexity. The paper investigates the use of the mentioned metrics to capture basic characteristics present in the multitasking systems. Additional extensions of both systems of metrics are proposed to increase the obtained level of information associated with the typical characteristics of multitasking systems.

The paper is organized as follows. Section 1 presents the analyzed Software Science metrics and the cyclomatic complexity metric. Section 2 contains a brief survey of proposed metrics for multitasking and concurrent software. Section 3 describes

a case study used to calculate the presented metrics. Discussion of obtained results in the context of chosen characteristics of the presented example is given in Section 4. Section 5 contains a proposal of extensions of both systems of metrics and their initial evaluation. Section 6 contains a conclusion and an outline of the future work.

1. Software Science and Cyclomatic Complexity Metrics

Halstead's Software Science theory defines the whole family of software complexity metrics [5], but the most commonly used Software Science metrics are *program length* N , *program vocabulary* n , *program volume* V and *predicted effort* E [4]. The mentioned metrics are calculated on the basis of the following program properties:

- n_1 – number of unique operators,
- n_2 – number of unique operands,
- N_1 – total number of operators,
- N_2 – total number of operands.

The program length N is defined as the sum of all occurrences of operators N_1 and operands N_2 in the program. The program vocabulary n is defined as the sum of unique operators n_1 and unique operands n_2 in the program. The program volume V corresponds to the volume of implementation measured as a number of bits required to encode the program and is calculated according to the formula:

$$V = N \log_2 n \quad (1)$$

The predicted effort E corresponds to a number of elementary mental discriminations performed by a skilled programmer during the implementation of a given algorithm and is calculated according to the following approximate formula [4]:

$$E = V (n_1 N_2) / (2 n_2) . \quad (2)$$

The main problem connected with the presented metrics is their insensibility to the level of complexity associated with the flow of control in a program. The problem has been addressed, among others, by Ramamurthy and Melton who proposed the synthesis of the Software Science metrics with the cyclomatic complexity [12].

McCabe's *cyclomatic complexity* metric is an attempt of use of the graph theory approach for the software complexity assessment [8]. The proposal is based on the *cyclomatic number*, which in the graph theory, is defined as a number of linearly independent paths of a strongly connected directed graph. The control flow graph G for the program's module can be constructed by the representation of all executable expressions as nodes and possible execution paths as edges of the graph. The resulting graph is not strongly connected, but it can be made so, through addition of a virtual edge between nodes representing the exit node and the entry node of the module. Each of program's modules p may be represented as a separate strongly connected graph. The control flow graph for the whole program is the union of all p strongly connected graphs. The cyclomatic complexity is calculated according to the formula:

$$v(G) = e - n + 2p \quad (3)$$

where e is a total number of edges in all graphs (excluding virtual edges), n is a total number of nodes and p is a number of modules in the program.

The main problem associated with the use of the cyclomatic complexity is its insensitivity to the software complexity connected with the program size. The problem has been addressed, among others, by Baker and Zweben in [2], Hansen in [6] and in the already mentioned proposal of Ramamurthy and Melton [12]. The other problem concerns treating complex conditional expressions as single conditional expressions or separate multiple conditional expressions. The problem has been addressed, among others, by Myers, who combined both mentioned approaches [7].

2. Complexity Metrics for Multitasking and Concurrent Software

Most of the existing theories concerning the software complexity assessment are oriented on measuring the complexity of sequential software. Since the multitasking and concurrent software has become widespread, some attempts have been made to address the issue of measuring the complexity of such type of software.¹

2.1. Existing Complexity Metrics for Multitasking and Concurrent Software

At the origin of research on the subject lays the work of Ramamoorthy et al. concerning the complexity of requirements [11]. In this work, the idea of dividing the complexity of concurrent program into two levels: sequential and concurrent complexity has been presented. Shatz applied a similar approach in [14] for an analysis of distributed programs written in the programming language Ada and has proposed the following formula describing the *total complexity TC* of the program:

$$TC = W \sum_{i=1}^T LC_i + X CC \quad (4)$$

where T is a number of tasks, LC_i is the *local complexity* of a task i (calculated with the use of standard metrics for sequential software), CC is the *concurrent complexity*, W and X are weight values. Shatz used Petri nets as a modeling tool and pointed nesting and independent rendezvous patterns as the primary factors influencing the complexity, but has not provided the final definition of the concurrent complexity. Zhenyu and Li continued the research for multitasking Ada programs and have proposed a definition of the concurrent complexity as a weighted sum of the *rendezvous successive complexity* and *nesting complexity* [16].

Tso et al. have made a different proposal for an analysis of avionics software written in Ada, but also based on the idea of separating sequential and concurrent program characteristics [15]. The approach was based on the Software Science metrics and divided operators and operands into two categories: sequential and so-called *Ada-*

¹ From the perspective of the software complexity analysis, most of general conclusions drawn from observations made for concurrent software also apply to multitasking software.

Multitasking-Realtime (AMR). The latter category contained Ada constructs used to support multitasking and real-time operations. In the presented approach, the Software Science metrics were calculated for combined numbers of operators and operands from both categories, but numbers for the AMR category were multiplied by an additional empirically evaluated weight value. A similar approach has been presented by Anger et al. for the analysis of the space shuttle software written in the HAL/S language [1]. The analysis included a set of standard metrics and an additional group of 10 metrics based on so-called *Real Time Executive* language constructs.

De Paoli and Morasca have presented a different approach [3], in which the idea of dividing the complexity of concurrent programs into two separate kinds of complexity was questioned. The proposal, as in case of Shatz's work, was also based on Petri net models, but the proposed metric, *concurrent complexity*, was calculated for the representation of a multitasking program as an extension of the cyclomatic complexity.

2.2. Limitations of Presented Approach

The approach presented in this paper is based on the results of the efforts mentioned in Section 2.1, but differs in a few significant points. The presented analysis is performed for software implemented in the programming language C, which does not provide any inherent support for the synchronization and communication mechanisms. The possibility of use of various architectural solutions to support operations typical for multitasking systems makes an automated analysis of the characteristics of such software a very difficult task. For example, construction of a Petri net representation of a set of tasks in such system may require so extensive knowledge of the system requirements, design and details of underlying software and hardware platform that the rationale of the idea of automated software analysis could be questioned.

The presented analysis focuses on increasing the level of information associated with multitasking characteristics of software, which is captured by already existing and widely utilized software complexity metrics. The proposed extended metrics are calculated in a similar way to the existing metrics, which allows to use already developed tools for the software complexity analysis.

Since the scope of work presented in the paper includes only an initial proposal of a new approach, a connection of obtained results with a subjective perception of software by software engineers is emphasized. A thorough statistical analysis of real-life multitasking systems is required to prove the usefulness of the presented approach.

3. Case Study: Simple Communication Scenario

To capture typical characteristics of multitasking software, a simplified hypothetical communication scenario has been implemented in three versions. Each next version is built on the basis of the previous one and differs in general complexity connected with the reliability of presented communication scheme. Because of reasons explained in Section 2.2, a single task, the *manager*, has been chosen for further analysis.

The proposed system consists of 4 tasks. The *requestor* task represents an external source of requests handled by the system. The *manager* coordinates the process of allocation and activation of resources. The *allocator* is responsible for the allocation of resources and the *executor* is responsible for the activation of resources.

In the basic version presented in Figure 1, 8 messages with self-explanatory names are exchanged between tasks. The communication scheme in this version considers neither the possibility of unsuccessful resource allocation or deallocation, nor the communication failure due to problems with the communication mechanism itself or the operation of tasks. Therefore, the presented scenario seems to have little or no resemblance to typical patterns or characteristics of real-life multitasking systems.

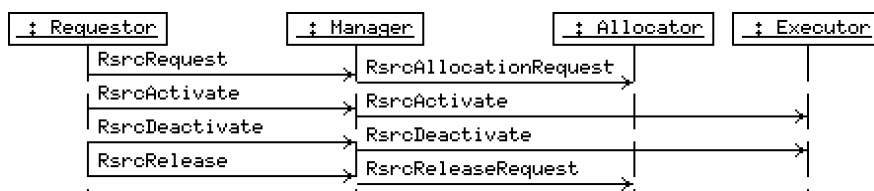


Figure 1. Basic version of the communication scenario

Figure 2 presents the second version of the communication scenario, in which 4 additional messages have been added to incorporate a mechanism of acknowledge during the allocation and deallocation of resources. Use of the mechanism of acknowledge may be considered a common practice in multitasking systems and definitely increases the reliability in the described scenario. Both the *manager* task and the *requestor* task are informed whether the operation is successful or not, and may alter their behavior appropriately to the situation.

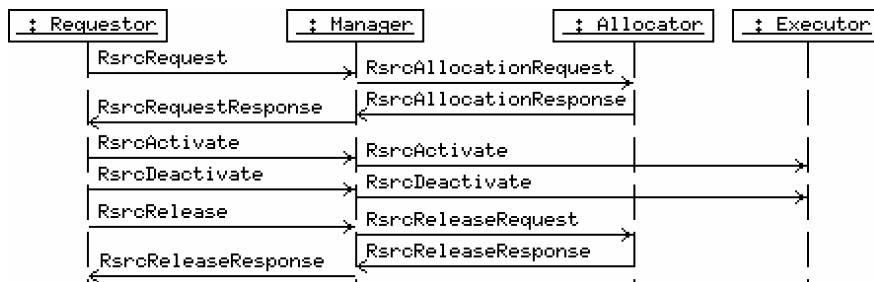


Figure 2. Second version of the scenario with the acknowledge mechanism

Figure 3 presents the third version of the communication scenario, in which the previously added acknowledge mechanism is additionally guarded by timers. Timers, from the perspective of a task, are set and canceled synchronously via function calls and the timer's expiry is connected with receiving of 2 additional messages not depicted in Figure 3. Addition of timers guarding the acknowledge messages further increases the reliability of the communication scenario and makes the third version of the scenario closer to patterns employed in real-life multitasking systems. Use of timers assures that the *manager* task shall respond to the *requestor* task within the specified period of time, even if the response from the *allocator* task has not been received.

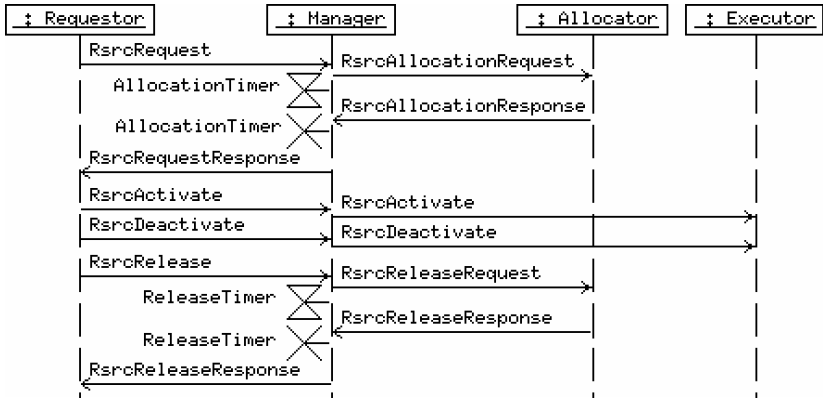


Figure 3. Third version of the scenario with the mechanism of acknowledge guarded by timers

4. Metrics for Sequential Software and Characteristics of Multitasking Software

The case study presented in Section 3 has shown some of the typical characteristics of multitasking systems, which are the use of the acknowledge mechanism and timers. The above characteristics represent only a subset of properties of multitasking software.² Because of reasons explained in Section 2.2, as the basis of the analysis presented in this section and Section 5, authors of the paper have initially focused on the following simplified indicators of mentioned characteristics:

- increase in a number of processed messages in case of the acknowledge mechanism,
- increase in a number of processed messages and presence of calls to timer related functions in case of the use of timers.

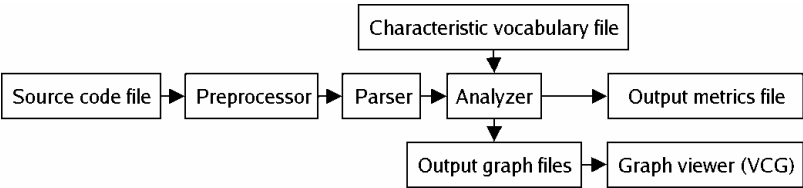


Figure 4. Environment used to calculate metrics and to generate control flow graphs

Figure 4 presents the environment of used tools. Metrics are calculated in the *analyzer* module on the basis of intermediate data collected during parsing of preprocessed *source code file* at the *parser* module (the *characteristic vocabulary file* is used for the calculation of extended metrics proposed in Section 5). As results of processing,

² Studies of various characteristics of multitasking and concurrent software may be found in references presented in Section 2.1

the *output metrics file* and the *output graph files* are generated. The graph files may be visualized with the use of the *VCG* tool [13].

In order to calculate the cyclomatic complexity, Formula 3 has been modified to account for the *additional complexity a* associated with complex conditions present within single executable statements:

$$V(G) = e - n + a + 2p \tag{5}$$

The Software Science metrics have been calculated according to description from Section 1. The results, which have been calculated separately for the main task routine and the whole task, for all versions described in Section 3, are presented in Table 1.

Table 1. Results of calculation of the Software Science and cyclomatic complexity metrics for all versions of the *Manager* task (the first column contains results for the task’s main routine, the second column contains results for the whole task).

Metric	Version 1		Version 2		Version 3	
N_1	44	286	52	388	70	510
n_1	26	42	28	46	30	51
N_2	50	397	58	545	78	775
n_2	32	96	36	114	41	148
N	94	683	110	933	148	1265
n	58	138	64	160	71	199
V	550,65	4855,12	660,00	6831,36	910,16	9660,33
E	11185,10	421637,02	14886,70	751149,86	25972,90	1256658,83
e	35	160	43	212	51	282
n	30	159	36	211	42	275
a	1	1	1	1	1	1
p	1	13	1	17	1	19
$v(G)$	8	28	10	36	12	46

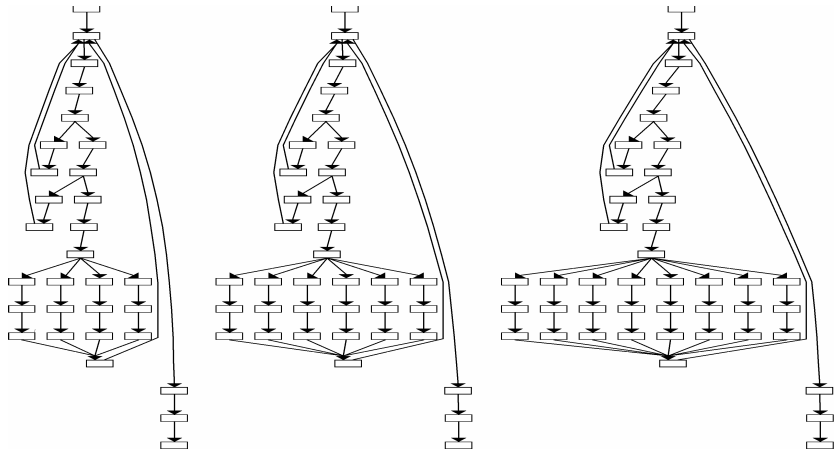


Figure 5. Control flow graphs of the main task routine of the *manager* task in version 1, 2 and 3 (from the left to the right).

On the basis of collected results, it can be observed that values of the Software Science metrics and the cyclomatic complexity increase with each version of the presented scenario. A question that could be asked is whether the increase in values of the metrics corresponds to our expectations connected with changing characteristics of the presented system. For that purpose, a numerical analysis of the correlation between numbers of processed messages and, or calls to timer related functions, might be performed. The problem is that obtained results would be misleading because of the two reasons.

The first reason is connected with the fact that analyzed system is an oversimplified example of a multitasking system. All presented tasks act according to the diagrams from Section 3, but they do not really perform any additional computations, maintain any information about their internal state or perform recovery procedures in erroneous situations. These are the elements that would be typically found in real-life implementations of multitasking systems. Therefore, the question could be: would it make sense to perform such analysis for real-life systems? The answer to that question is, again, rather negative because of the second of mentioned reasons.

The second reason is much more severe and is connected with the fact that the presented metrics have been created mainly for sequential software. It means that they are not oriented on capturing any additional software characteristics than those that are inherent in their nature. Software Science metrics are based on numbers of operators and operands, and only on those properties of the program code. The cyclomatic complexity is associated only with the flow of control in the program, or to put in other words, with the presence of language constructs modifying the flow of control.

The above statements may be easily illustrated by the following examples. Adding an identifier representing a new message recognized in the task's main routine has a comparable impact on numbers of operands and unique operands as adding an ordinary variable in one of the task's helper functions. Adding an additional `case` clause for a new message processed by the task in the `switch` statement of the main task's loop has a comparable impact on the cyclomatic complexity as an input parameter check in one of the helper functions. For the cyclomatic complexity, the presented problem can be easily observed on control flow graphs of the main task routine, which have been presented in Figure 5. Results in Table 1 and the presented graphs show that with addition of each new incoming message recognized by the task, the cyclomatic complexity of the main task routine increases by one. However, such increase in the complexity has exactly the same weight as in case of all other statements changing the flow of control within the main task routine and in the whole program.

5. Complexity Metrics Extensions for Multitasking Systems

As it has been shown in Section 4, both the Software Science metrics and the cyclomatic complexity do not capture chosen characteristics of multitasking systems or to be more precise, the language constructs associated with specific characteristics of multitasking software are recorded as all other similar constructs in the program. A modification of both metrics is required to extract the desirable characteristics.

5.1. Proposal of Complexity Metrics Extensions for Multitasking Systems

In case of the Software Science metrics, a slightly modified approach presented by Tso et al. in [15] can be applied. For that purpose, a set of items associated with chosen characteristics of analyzed software has to be identified. In the presented solution, the set of items, called a *characteristic vocabulary*, consists of:

- numerical identifiers of messages exchanged between the tasks,
- name of a function used to receive messages from the task's message queue,
- name of a function used to send messages to other tasks' queues,
- names of functions used to set and cancel timers.

Each operator or operand that is found in the characteristic vocabulary is classified as a *characteristic operator* or *characteristic operand*.³ Such classification allows to specify the following input parameters:

- n_{1c} – number of unique characteristic operators,
- n_{2c} – number of unique characteristic operands,
- N_{1c} – total number of characteristic operators,
- N_{2c} – total number of characteristic operands,

for the process of calculation of the Software Science described in Section 1. To distinguish new values of Software Science metrics from the standard ones a prefix *characteristic* is added for each of the metrics. The results are presented in Table 2.

In case of the cyclomatic complexity, an approach similar to the proposed by McCabe *design complexity* [9], and *specified data complexity* [10], can be applied. The main idea standing behind the calculation of both metrics is to specify nodes in the control flow graph that have a certain property and to apply a set of rules presented in Figure 6 in order to reduce the input control flow graph. The value of the metric is equal to the cyclomatic complexity calculated for such reduced graph.

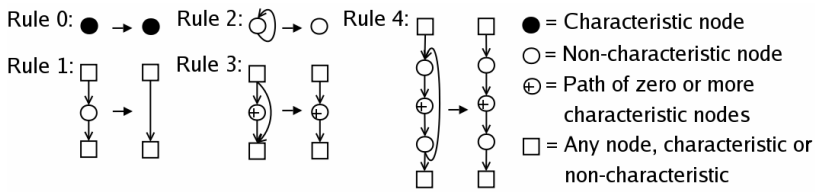


Figure 6. Graph reduction rules proposed by McCabe [9]

In the presented approach, all nodes that contain items from the previously defined characteristic vocabulary are treated as *characteristic nodes*. The results obtained for the proposed *characteristic cyclomatic complexity* are presented in Table 2.

³ There is no need to create separate sets of items for operators and operands, because during the parsing process function names are classified both as operators and operands, and all remaining identifiers, constants and variables are classified as operands.

Table 2. Results of calculation of extended Software Science metrics and cyclomatic complexity for all versions of the manager task (the first column contains results for the task’s main routine, the second column contains results for the whole task).

Metric	Version 1		Version 2		Version 3	
N_{lc}	1	5	1	7	1	11
n_{lc}	1	2	1	2	1	4
N_{2c}	5	13	7	19	9	27
n_{2c}	5	10	7	14	9	18
N_c	6	18	8	26	10	38
n_c	6	12	8	16	10	22
V_c	15,51	64,53	24,00	104,00	33,22	169,46
E_c	7,75	83,89	12,00	141,14	16,61	508,38
e_c	13	33	17	45	21	55
n_c	10	42	12	56	14	66
a_c	0	0	0	0	0	0
p_c	1	13	1	17	1	19
$v_c(G)$	5	17	7	23	9	27

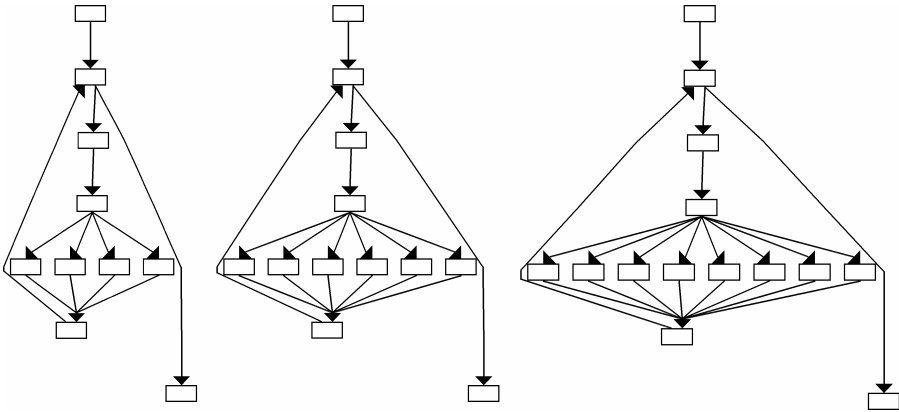


Figure 7. Reduced control flow graphs of the main task routine of the manager task in versions 1, 2 and 3 (from the left to the right).

5.2. Initial Evaluation of Proposed Metrics

For the extended Software Science metrics, the interpretation of the results seems to be the most meaningful on the task level, on which the following observations concerning the input parameters can be made:

- total number of characteristic operators N_{lc} is equal to the sum of one call to the function retrieving messages from the task’s queue, a number of calls to the functions sending outgoing messages and a number of calls to timer related functions,
- number of unique characteristic operators n_{lc} is equal to a number of distinct functions from the characteristic vocabulary called in the task’s code,

- total number of characteristic operands N_{2c} includes the total number of characteristic operators N_{1c} plus the sum of all occurrences of message identifiers (in case of the version 3 of the presented scenario, there are 2 additional occurrences of timeout messages identifiers passed as parameters to the function setting timers),
- number of unique characteristic operands n_{2c} includes the number of unique characteristic operators n_{1c} plus the sum of all message identifiers.

It can be observed that values of metrics increase with the increase of general complexity of the communication pattern, but although all input parameters are directly connected with the multitasking characteristics pointed in Section 4, it is difficult to give the appropriate interpretation of the extended Software Science metrics through an analogy to the original description of the Halstead's Software Science metrics. For example, in case of the characteristic predicted effort E_c , by making a similar assumption as in the Halstead's work [4], that a skilled programmer makes about 18 elementary mental discriminations per second, the period of time required to understand the communication pattern of the most complex version of the presented example by looking at its code would be about half a minute. The result is probably closer to the period of time required to understand the presented communication scenario by looking at the diagram on Figure 3. Still, it seems that the proposed metric may indeed have some connection with a mental effort required to understand the presented communication pattern, but its usefulness, as in case of remaining metrics, has to be proved during the empirical verification.

Interpretation of the characteristic cyclomatic complexity metric is much easier as the subject has been already discussed by McCabe for the design complexity and specified data complexity metrics. According to McCabe [4], [5], a value of the cyclomatic complexity calculated for reduced graph is equal to a number of independent paths in the program, which need to be executed during the integration tests, in case of the design complexity, and for example, during regression tests after the modification of code using the specified data, in case of the specified data complexity. A similar interpretation may be given for the proposed characteristic cyclomatic complexity. The results presented in Table 2 show that along with the increase of general complexity of the communication pattern, it is the complexity of the main task routine and a number of modules of the task that contributes to the overall value of the metric for the program. The characteristic cyclomatic complexity of the main task routine increases by a number of distinct incoming messages recognized by the task (see Table 2 and in Figure 7). All other modules are reduced to single execution paths, for which the characteristic cyclomatic complexity equal to one. By an analogy to the already presented interpretation, each of the independent paths of the main task routine should be executed during tests of the communication mechanisms, while it should be enough to enter each of remaining modules of the task only once.

The results obtained with the use of the proposed characteristic cyclomatic complexity are very encouraging. It should be noticed though, that the information connected with calls to timer related functions, which have been placed in message handler routines with a relatively simple flow of control, was lost during the reduction process. Such property of the reduction process suggests that, as in case of cyclomatic complexity, an additional perspective, possibly the one applied in the proposed characteristic Software Science metrics, should be used.

6. Conclusions and Future Work

In the paper, the use of two of the most widely utilized software complexity measurement theories: Halstead's Software Science metrics and McCabe's cyclomatic complexity, has been investigated for the purpose of an analysis of basic characteristics of multitasking systems implemented in the programming language C. Problems associated with the capability of the mentioned metrics to capture typical characteristics of such systems have been pointed out on the basis of a simple example of chosen communication patterns. Additional extensions of both systems of metrics have been proposed to increase the level of obtained information connected with the typical characteristics of multitasking systems.

The presented initial results are encouraging, but the usefulness of the proposed extended metrics needs to be verified during the statistical analysis of results collected for real-life examples of multitasking systems. The analysis of empirical results should allow to identify the appropriate interpretation for some of the proposed metrics as the interpretation analogical to the one connected with the definition of the source metrics for sequential software cannot be directly applied.

References

- [1] Anger, F.D., Munson, J.C., Rodriguez, R.V.: Temporal Complexity and Software Faults. *Proceedings of the 1994 IEEE International Symposium of Software Reliability Engineering* (1994) 115-125
- [2] Baker, A.,L., Zweben, S.H.: A comparison of measures of control flow complexity. *IEEE Transactions On Software Engineering*, vol. SE-6 (1980) 698-701
- [3] De Paoli, F., Morasca, S.: Extending Software Complexity Metrics To Concurrent Programs. *Proceedings of the 14th Annual International Computer Software and Applications Conference* (1990) 414-419
- [4] Fitzsimmons, A., Love, T.: A Review And Evaluation Of Software Science. *Computing Surveys*, Vol. 10, No. 1 (1978) 3-18
- [5] Halstead, M.H.: Elements of software science. *Elsevier North-Holland, Inc*, N Y (1977)
- [6] Hansen, W.: Measurement of program complexity by the pair (cyclomatic number, operator count). *SIGPLAN Notices*, 13, (3) (1978) 29-33
- [7] Myers, G.J.: An extension to the cyclomatic measure of program complexity. *SIGPLAN Notices*, 12, (10) (1977) 61-64
- [8] McCabe, T.J.: A complexity measure. *IEEE Transactions on Software Engineering*, vol. SE-2, No. 4 (1976) 308-320
- [9] McCabe, T.J., Butler C.W.: Desing Complexity Measurement and testing. *Communications of the ACM*, vol. 32, No. 12 (1989) 1415-1425
- [10] McCabe, T.: Cyclomatic complexity and the year 2000. *IEEE Software, Software Development*, Volume 6, Issue 7 (1996) 115-117
- [11] Ramamoorthy, C.V., Tsai, W.T., Yamaura, T., Bhide, A.: Metrics guided methodology. *Proceedings of 9th Computer Software and Application Conference* (1985) 111-120
- [12] Ramamurthy, B., Melton, A.: A synthesis of software science measures and the cyclomatic number. *IEEE Transactions On Software Engineering*, vol. 14, nr 8 (1988) 1116-1121
- [13] Sanders, G.: VCG, Visualization of Compiler Graphs. *User Documentation*, V.1.30 (1995)
- [14] Shatz, S.M. Towards Complexity Metrics for Ada Tasking. *IEEE Transactions on Software Engineering*, Vol. SE-14, No. 8 (1988) 1122-1127
- [15] Tso, K.S., Hecht, M., Littlejohn, K.: Complexity Metrics for Avionics Software, *Proc. of the National Aerospace and Electronics Conference* (1992) 603-609
- [16] Zhenyu, W., Li, C.: Ada Concurrent Complexity Metrics based on Rendezvous Relation. *Proceedings of the 18th Computer Software and Applications Conference* (1994) 133-138

4. Technologies for SOA

This page intentionally left blank

Grid Enabled Virtual Storage System Using Service Oriented Architecture

Łukasz SKITAŁ^a, Renata SŁOTA^a, Darin NIKOLOV^a
and Jacek KITOWSKI^{a,b}

^a *Institute of Computer Science, AGH-UST,
Al. Mickiewicza 30, Cracow, Poland*

^b *Academic Computer Center CYFRONET AGH,
ul. Nawojki 11, Cracow, Poland*

Abstract. Emerging of the web services technology gives Service Oriented Architecture a new practical meaning. The architecture allows developing of complex, distributed systems in a multi-platform environment. Recent Grid projects take an advantage of this architecture, as it is applicable to systems developed by a number of different institutes, computing centers and companies. One of the important aspects of the Grid projects is data management, which is the subject of our study.

In this paper we present our experience from designing and implementing Virtual Storage System in Service Oriented Architecture.

Introduction

The scientist's community becomes more aware of great scope of new technologies, like the Grid [1]. With this awareness their expectations are more defined and guide the further development of the grid technologies. Contemporary Grid can offer access to a variety of resources, like computational power, storage space or even scientific equipment. Scientists need an easy way to perform their experiments in efficient way. Therefore there are two main tracks in grid development: ease of use and high performance. One of the aspects, which has great impact on Grid system performance is efficient data access. Data size, used by computation tasks, e.g. simulation or visualization, range from megabytes to gigabytes or even terabytes. To assure efficient data access, the Grid environment has to be equipped with high performance, distributed data management system. Such environments called Data Grids have been a research topic in recent projects like Storage Resource Broker [2], European DataGrid [3], GriPhyN [4]. Growing demands and user expectations enforce the use of sophisticated data access optimization methods [5]. One of these optimization methods is data replication [6].

Service Oriented Architecture (SOA) is a software architectural concept that defines the use of services to support applications requirements. The concept behind the SOA is known over a decade, but recently it has taken more practical application. As a web services technology appeared, the SOA can be used easily in newly developed systems [7], [8]. Also it is possible to extend legacy software by new services, but this task needs some effort to make the software SOA enabled.

As WebServices or GridServices¹ have become Grid middleware implementation standards, there is an open way to the SOA. Indeed most of the recent Grid projects are using loose coupled services, which can be easily reused in new projects.

In the paper we present our experience from designing and implementing the Virtual Storage System (VSS) for the Grid environment using SOA. We have focused on describing the implementation process rather than discussing the underlying algorithms. Detail about the algorithms themselves can be found in [9]. The system has been implemented as a part of SGIGrid project [10].

The remaining of the paper is organized as follows: in Section 1 we take a closer look at SOA and present the goal of our study. Next, the requirements and properties of the VSS are discussed and the system architecture is shown to indicate the complexity of the system. In Section 3 we look over the system implementation taking into account SOA. In Section 4 exemplary performance tests are presented. In the last Section 5 conclusions about the usefulness of SOA in the implementation of VSS are presented.

1. Background

The SOA is an attempt to solve most of the weak points of object oriented and component programming [11]. It allows for seamless multi-platform integration and straight forward parallel development. The problem of rapid system composition is addressed by many software vendors [13], [14], [15]. Development platforms for SOA usually use BPEL² with some graphic user interface, which allows system composition using “drag & drop” method. Different approach is proposed in [16], where services are automatically composed using some knowledge base.

The SOA with web services technology can significantly speedup development of complex, distributed systems. Every service is responsible for some functionalities and can be developed independently from the rest of the system. As long as the service interface syntax and semantics are not changed, a new version of the service can be deployed with no need to modify, recompile or even restart the remaining system services. One of the SOA paradigm is the “loose coupling” of the services [12]. In practice it means that the services should offer some complete functionalities which can be reused apart from the rest of the system. Another feature of SOA is defining the use of services (workflow) for the given process execution.

The web services technology do not enforce SOA. There are many systems, that were developed using WebServices, but they are not real SOA. These systems usually have strong inter-service dependency – the functionality of these services can not be used without the rest of the system. However, these systems could take great advantage of the SOA. They are build up from web services, so they are compatible with other services and can be easily enhanced by reuse of existing services. This leads to a kind of hybrid system, where there is a core system – not reusable, with strong dependencies – and the set of loose coupled services offering some new functionalities. This case is discussed in the paper.

The SGI Grid project [10] aims to design and implement broadband services for remote access to expensive laboratory equipment, backup computational center and remote data-visualization service. Virtual Laboratory (VLab) experiments and visuali-

¹ The expanded version of WebServices, developed as a part of the Globus Toolkit 3

² Business Process Execution Language

zation tasks very often produce huge amount of data, which have to be stored or archived. The data storage aspects are addressed by Virtual Storage System (VSS). Its main goal is to integrate storage resources distributed among computational centers into common system and to provide storage service for all SGI Grid applications. The Virtual Laboratory, part of the SGI Grid, choose the Data Management System (DMS)³ as a storage system. Thus, to preserve the system uniformity, DMS has been chosen as the base system for VSS (see Section 2.2). The implementing of VSS based on DMS, built by tightly coupled web services, become an opportunity to study the usefulness of the SOA paradigm.

2. Virtual Storage System

2.1. Properties and Requirements

Data management systems usually possess basic functionality like the ability to store and retrieve files organized in a common virtual file system. Virtual Storage System (VSS) for the SGIGrid project has some set of special requirements.

HSM Support

HSM based Storage Elements (SE) are used to store large amount of data which are rarely accessed by users, making up a kind of archive. There is a need to distinguish between HSM based SE and hard disk based SE. VSS users can assign *immediate* or *archived* attribute to any file. The *immediate* attribute means, that the file should be stored on SE with short access time. The *archived* attribute means that the file should be stored on HSM. If no attribute is assigned to the file, the system will select SE according to the file size and available storage capacity, “big” and rarely used files are placed on HSM based SE and “small” on others SE.

File Ordering

The access time for files stored on HSM based SE can reach values of up to tens of minutes. In order to shorten the access time our VSS provides *file ordering mechanism* being a kind of method for prefetching scheduling. The file can be ordered for a specified time interval. The system will copy the file from tape to a disk cache, so that at the specified time, the file can be accessed immediately.

File Fragment Access

A file fragment can be also ordered. This functionality is very useful for retrieving specific data from a large file. The knowledge about file structure is necessary to pinpoint a proper file fragment (as a byte range). This operation limits the time needed to access the file fragment thanks to the support of low level middleware for HSM systems.

Automatic File Replication

This is a process of making additional copies of files in order to shorten the access time. These copies (replicas) can be used in the same manner as the original file. The decision about a replica creation is based on past data usage.

³ The DMS is developed at Poznan Supercomputing and Networking Center

Optimal Replica Selection

An algorithm for selection of optimal replica is necessary to achieve some gain from data replication. The optimal replica selection is made based on the estimated data access time of the storage element itself and the measured network latency between the storage elements and the destination.

Storage Access Time Estimation

The implementation of any replica selection algorithm requires information about the storage access time of the requested files. The storage access can be defined as a time from the moment of file request till the moment of sending of the first byte. An important issue is access time estimation of data residing on HSM systems. The access time estimation system for the DiskXtender [17], [18] developed as part of our previous study has been used for that purpose.

Replica Coherency

The VSS implements the strong coherency model – only the newest file version can be accessed. This is possible to achieve thanks to centralized metadata repository. Older versions of the file are successively updated.

2.2. The Base Software for VSS

The Data Management System (DMS), developed by the Progress project [19], has been chosen as a base system for the VSS in the SGIGrid project.

The DMS provides a uniform interface to distributed data storage i.e. to storage elements (SE). All data files are organized in a virtual file system. The user willing to store or retrieve data is provided with URL, which points directly to the data storage. FTP and GridFTP protocols can be used to transfer the data. The DMS supports manual replication: the user is responsible for replica creation or removal and actually has to perform the data transfer.

The DMS is implemented in the Java language using web services, but it is not the real SOA. The dependencies between the system services are strong, thus they can not be used separately.

To provide functionalities required by the VSS, the existing DMS services has been modified and the new services has been added.

2.3. VSS Architecture

The VSS is implemented in the Java language and uses the web service technology. The SOAP protocol [20] is used for communication between the system services. GridFTP and FTP protocols are used for data transfer. We use the GridFTP server provided with the Globus Toolkit.

The system architecture is shown in Figure 1. Base DMS system is represented by white boxes. Gray boxes represent services (or group of services) which have been implemented and added to DMS to fulfill the requirements for VSS.

The system consists of the following services:

- **Data Broker** – the service responsible for authentication and authorization of the users,

- **Metadata Repository (MR), Log Analyzer (LA)** – set of the system core services, responsible for every aspect of the system operation. MR consists of: *directory service* – responsible for virtual file system structure, *storage service* – responsible for file operations and virtual names to physical locations mapping, *metadata service* – responsible for data description. Log Analyzer is part of the storage service. It is responsible for gathering data about the previous data access patterns, which are used by the Replica Manager.
- **Data Container** – consists of service for physical file storage management and file transfer via HTTP, GridFTP and FTP. The service is responsible for data storage and sharing.
- **Data Container Extension (DCE)** – provides a set of services with new functionalities. Consists of the file ordering and file fragment access service, storage access time estimation service, file transfer service.
- **Replica Manager (RM)** – is the service responsible for the automatic creation and deletion of replicas. It cooperates with the MR to assure replica coherency.

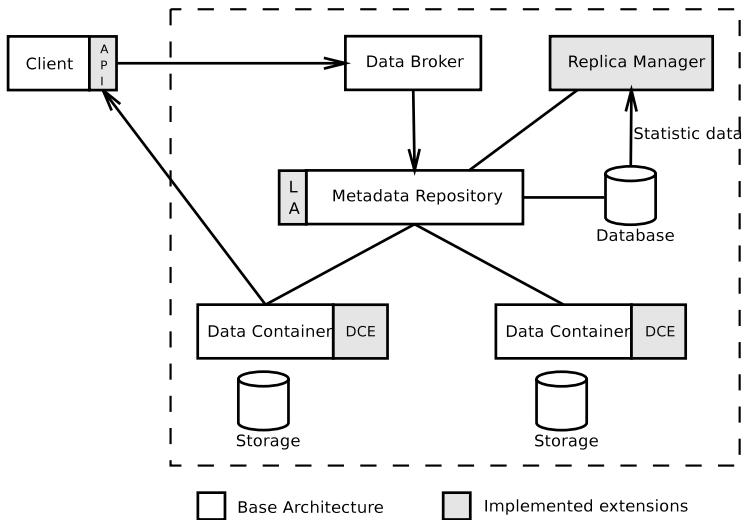


Figure 1. Architecture of the VSS

The base system services (white boxes) are tightly coupled, i.e. they can not be separated from the system and reused apart from it without vast code modifications. The new services (RM, DCE) are loosely coupled. Their functionality can be reused.

3. Implementation Overview

The implementation of the VSS required creation of new services (RM, DCE), as well as modifications of the existing ones.

The following sections will discuss in detail the implementation of new functionalities across the existing and new system services.

3.1. Storage Access Time Estimation

The storage access time estimation service provides access to an external estimator. Estimator is a legacy program, which returns estimated storage access time for a given physical file stored on the underlying storage system.

As a part of previous studies, an estimator for the DiskXtender HSM system has been developed and used. In case of using other storage systems, an appropriate estimator should be provided by the system administrator.

The storage access time estimation is implemented as a part of the DCE in a form of new service. However, in order to provide this functionality to the users, an extension of the interface of the Data Broker was necessary, as well as addition of a new method to the MR. The method gets all locations of a file and asks each location's storage access time estimation service for the estimated storage access time of that file location. The existing methods have not been modified.

In Figure 2 the UML sequence diagram for Storage Access Time Estimation is shown.

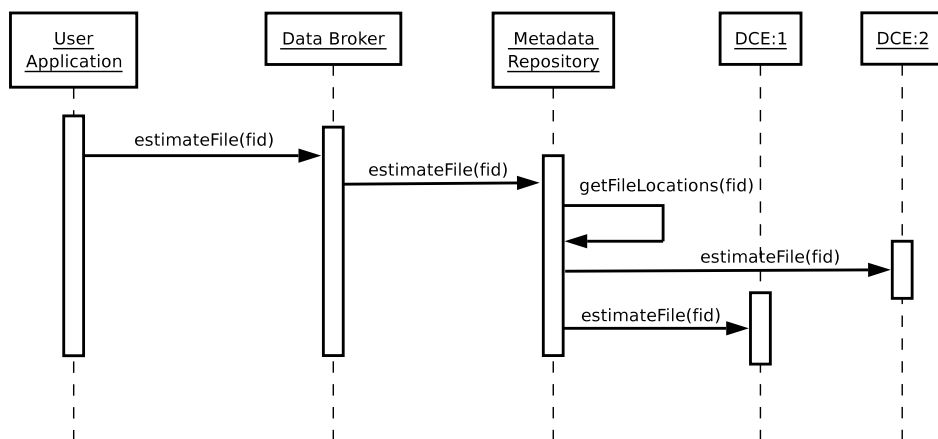


Figure 2. Storage Access Time Estimation sequence diagram

3.2. File Ordering

Similarly to the storage access time estimation, the file ordering functionality is part of the DCE. It is represented by a newly defined service. Also this time, apart from the extension of the Data Broker's interface, a new method in the Metadata Repository had to be implemented, but no modifications to the existing methods were necessary. The new method is responsible for the selection of an optimal location for a file order and forwarding the file order to that location.

The file ordering service for each new file order computes the time, when to start file transfer from tapes to disk cache. The time is computed according to the user requirement given in the file order and the estimated storage access time (with some safety margin). When the operation of file caching is finished, the file is locked in cache, till the user specified moment of requesting. After that moment the lock is

removed no matter if the file has actually been accessed. The lock removal allows the file to be removed, if necessary, by the HSM system according to its purging policy.

In Figure 3 the UML sequence diagram for File Order is shown.

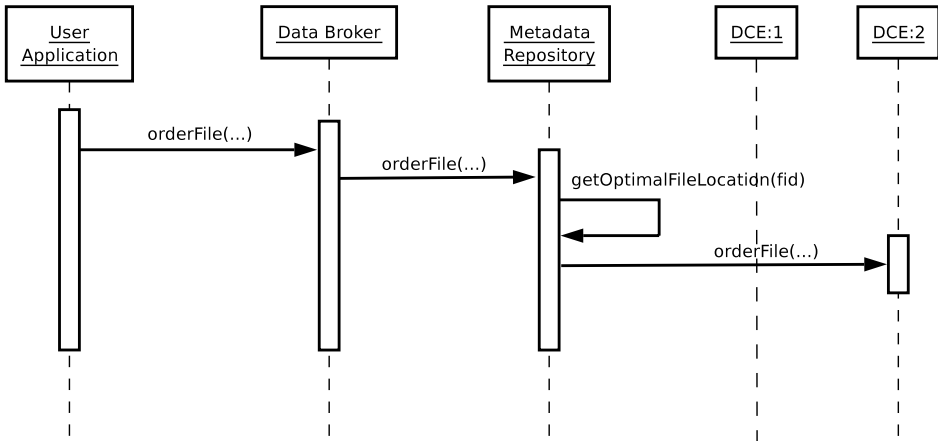


Figure 3. File Ordering sequence diagram

3.3. Access to File Fragments

The access to file fragments is implemented in the file ordering service. The user can specify a file fragment by defining its byte range. The access to file fragments is done in the same way as the file ordering. The only difference is, that the system (more precisely file ordering service) is not copying the whole file to the cache but only the requested fragment. When the fragment is no longer needed it is removed from the cache.

The presented above functionalities – Storage Access Time Estimation, File Ordering and Access to File Fragments – can be easily reused, they do not depend on any other system service and are implemented as web services grouped in DCE.

3.4. Replica Selection Algorithm

The algorithm is an integral part of the Metadata Repository, so the modification of the existing code was necessary. The existing, old algorithm has been entirely replaced by the new one. The new algorithm chooses the replica with the minimal access time, which is estimated based on the current system condition (storage system load and performance, network latency).

3.5. Automatic Replication

The automatic replication is implemented as a new service – RM. The service periodically checks the file access logs and evaluates decisions about replication.

For supporting the automatic replication, new specific methods have been added to the MR. These methods are used for selection of the optimal location for a new file and for the addition of a new replica to the repository.

In order to perform file transfers, the adequate file transfer service had to be implemented. The service was implemented as a part of the DCE. It allows to upload and download files from or to the SE. Actually only the FTP protocol is supported.

The Replica Manager works accordingly to the algorithm, which outline is presented below:

1. get the file access logs,
2. logs analysis – selection of files, which need to be replicated,
3. for each selected file:
 - a) get optimal⁴ storage element for new file,
 - b) get optimal⁵ source file location,
 - c) transfer the file using the DCE file transfer service,
 - d) add new replica to the repository.

In order to get the file access logs, Replica Manager executes a script provided by the system administrator. The script should place the logs in a proper directory, which is set in the RM configuration file. In the case, when RM is deployed on the same machine as the Metadata Repository, the script can simply copy the files or do nothing (depending on configuration). In the other cases, it should transfer the logs using appropriate file transfer protocol.

Automatic replication UML sequence diagram is shown in Figure 4.

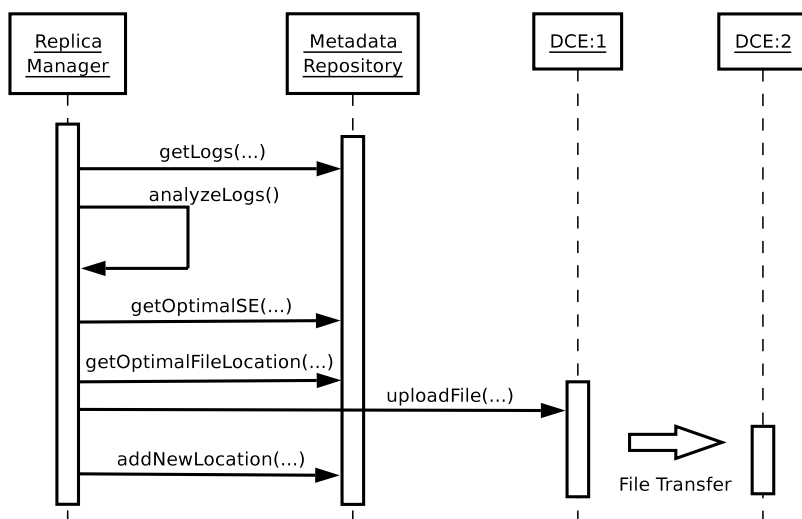


Figure 4. Automatic Replication sequence diagram

⁴ The optimal storage element is selected based on its storage class, storage system load and performance, network latency

⁵ The optimal source file location is the location with the minimum access time for the selected storage element

3.6. Replica Coherency

Maintaining replica coherency is a complex task that requires cooperation between Metadata Repository and Replica Manager. MR is responsible for preserving file consistency, while RM controls replicas propagation process.

MR as a basic DMS service has the ability to preserve file consistency by keeping the replicated files read-only. In order to allow replicated file updates, the replica coherency algorithm has been implemented in MR. In the case of replicated file update MR marks all replicas as *stale* and allows the user to update one replica only. While updated, MR allows access to the updated replica and blocks access to the remaining replicas. MR sends the list of the stale replicas and the location of the updated replica to RM.

During the replica propagation process, RM sends file transfer requests to appropriate file transfer service. After every successful upload, MR is notified and allows access to the replica just uploaded.

The cooperation of RM and MR in process of replica update is shown in Figure 5.

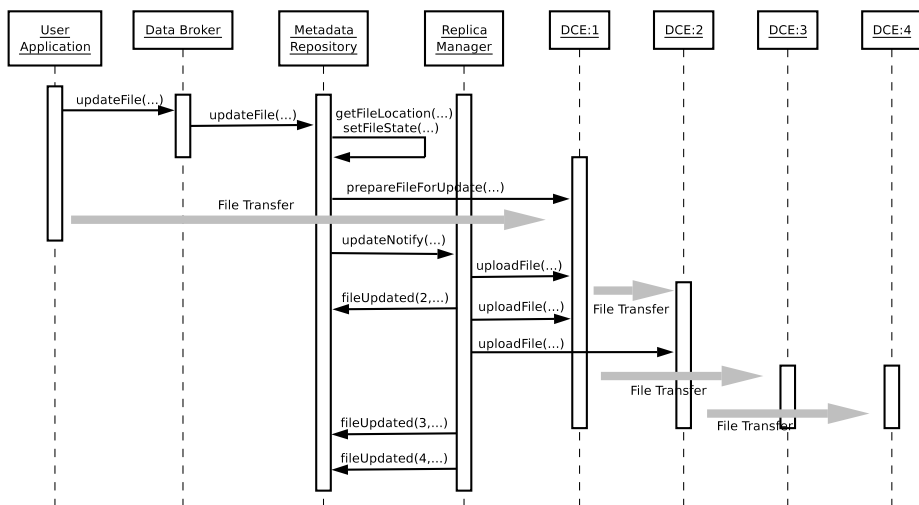


Figure 5. Replicas Update sequence diagram

4. Exemplary Performance Test Results

VSS has been tested by a set of validation and performance tests. Exemplary performance test results are presented in this section.

The VSS during the tests consisted of data containers localized in various sites in Poland. In the performance test, there were set of files having different sizes (10-1000MB). The client requests have been simplified to read-whole-file requests. The clients were located at different sites. Every client performed 100 random file reads before replication, and 100 file reads after replication. In both cases the same random pattern has been used and the access times for these files have been measured. By

access time to a file we assume the time from the moment of issuing a request till the moment when the file is completely transferred.

Test results for medium files are presented in Figure 6. The α coefficient is the parameter of the Zip-like random distribution used in the test. Significant performance gain has been possible due to different performance of the storage containers. All tests have been performed in low loaded environment. The increase of the storage usage after replication was about 50%.

For detailed description of the tested system, procedures and more test results see [9].

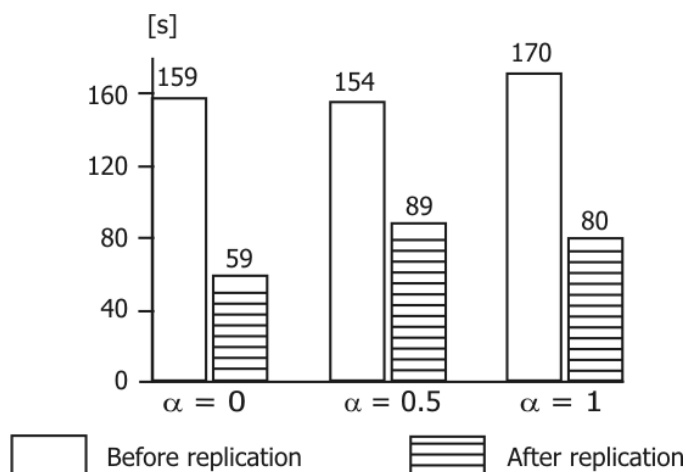


Figure 6. Average Access Time – medium files (100-300Mb)

5. Conclusions

The presented overview shows a way of implementation of Virtual Storage System within the framework of existing data management system. Thanks to the web service technology and the Java object programming language, implementation of VSS based on DMS has been possible without major modification of the existing DMS code. SOA has been used as the most applicable way of extending system's functionalities. Most of the VSS new features have been implemented as new services, which are optional for basic DMS. Nevertheless, the implementation of automatic replication functionality required modification of the basic DMS services. The basic DMS replica selection algorithm has been replaced with more advanced one. Also, in order to support replica updates, a new replica coherency algorithm has been added. Due to the flexibility offered by SOA it is possible to deploy the system with desired set of the services, despite the basic DMS services modifications.

The Virtual Laboratory (VLab), the part of SGIGrid project, is also based on DMS. Due to the SOA, the modification of the existing DMS code was minimized, during the VSS and VLab implementation. This allowed for parallel development of both DMS based systems.

Acknowledgments

The work described in this paper was supported by the Polish Committee for Scientific Research (KBN) project “SGIGrid” 6 T11 0052 2002 C/05836 and by AGH grant.

Thanks go to our colleagues from the Poznań Supercomputing and Networking Center and the Computer Center of Technical University of Łódź for cooperation.

References

- [1] Foster, I., Kesselman, C. The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition, *Morgan Kaufmann*, 2004.
- [2] Storage Resource Broker, <http://www.sdsc.edu/srb/>
- [3] DataGrid – Research and Technological Development for an International Data Grid, EU Project IST-2000-25182.
- [4] Grid Physics Network <http://www.griphyn.org/>
- [5] Dutka, Ł., Słota, R., Nikolow, D., Kitowski, J., Optimization of Data Access for Grid Environment, in: Rivera, F.F., et al. (Eds.), *Proceedings of Grid Computing, First European Across Grids Conference*, Santiago de Compostela, Spain, February 2003, Lecture Notes in Computer Science, no. 2970, Springer, 2004, pp. 93-102.
- [6] Lamehamedi, H., Szymanski, B., Deelman, E., Data Replication Strategies in Grid Environments, *Proceedings of the 5th International Conference on Algorithms and Architectures for Parallel Processing*, ICA3PP2002, Beijing, China, October 2002, IEEE Computer Science Press, Los Alamitos, CA, 2002, pp. 378-383.
- [7] Web Service and Service Oriented Architecture <http://www.service-architecture.com/>
- [8] Borges, B., Holley K., Arsanjani, A., Service-oriented architecture, 15 September 2004. <http://SearchWebServices.com>
- [9] Słota, R., Nikolow, D., Skitał Ł., Kitowski, J., Implementation of replication methods in the Grid Environment, to appear in: Proc. of the European Grid Conference, February 14 -16 2005, *Science Park Amsterdam*, The Netherlands
- [10] SGIGrid: Large-scale computing and visualization for virtual laboratory using SGI cluster (in Polish), KBN Project, <http://www.wcss.wroc.pl/pb/sigrid/>
- [11] Hashimi, S., Service-Oriented Architecture Explained, 18 Aug 2003, O'Reilly ONDotnet.com http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html
- [12] He H., “What is Service-Oriented Architecture?” O'Reilly webservices.xml.com, September 30, 2003 <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>
- [13] Oracle BPEL Process Manager <http://www.oracle.com/technology/products/ias/bpel/index.html>
- [14] SAP NetWeaver: Providing the Foundation to Enable and Manage Change <http://www.sap.com/solutions/netweaver/index.epx>
- [15] IBM WebSphere Business Integration Server <http://www-306.ibm.com/software/integration/wbiserver/>
- [16] The Knowledge-based Workflow System for Grid Applications (K-Wf Grid) <http://www.kwfgrid.net/main.asp>
- [17] Legato Systems, Inc. – DiskXtender Unix/Linux, <http://www.legato.com/products/diskxtender/diskxtenderunix.cfm>
- [18] Stockinger, K., Stockinger, H., Dutka, Ł., Słota, R., Nikolow, D., Kitowski, J., Access Cost Estimation for Unified Grid Storage Systems, *4-th International Workshop on Grid Computing* (Grid 2003), Phoenix, Arizona, November 17, 2003, *IEEE Computer Society Press*
- [19] Polish Research On Grid Environment for Sun Servers, <http://progress.psn.pl/>
- [20] WebServices – SOAP <http://ws.apache.org/soap/>

Internet Laboratory Instructions for Advanced Software Engineering Course¹

Ilona BLUEMKE and Anna DEREZIŃSKA
*Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/17, 00-665 Warsaw, Poland
e-mails: {I.Bluemke, A.Derezinska}@ii.pw.edu.pl*

Abstract. In this paper a new software engineering laboratory introduced in the Institute of Computer Science Warsaw University of Technology in the fall 2004 is presented. Advanced Software Engineering 2 (SE-2) laboratory consists of seven exercises. These exercises are dedicated to requirements engineering, system design with UML [11], reuse, precise modelling with OCL – Object Constraint Language [12], code coverage testing, memory leaks detection and improving application efficiency. Six out of ten SWEBOK [4] knowledge areas are practiced. For each laboratory exercise a set of training materials and instructions were developed. These materials are stored on a department server and are available for all students and lecturers of advanced Software Engineering 2 (SE-2) course. Rational Suite tools are used in laboratory.

Introduction

In the spring semester of 2004, a new advanced software engineering course, named Software Engineering 2 (SE-2), was introduced for students on the sixth semester of two specializations i.e. Engineering of Information Systems in the Institute of Computer Science, and Information and Decision Systems in the Institute of Control and Computation Engineering at the Department of Electronics and Information Technology, Warsaw University of Technology. This course contains 30 hours of lectures and 15 hours of laboratory exercises. SE-2 is a continuation of Software Engineering (SE) course given to the same students on the fifth semester. During Software Engineering course semester, students learn the basic ideas of software engineering and object modelling in UML (Unified Modelling Language [11]). They also spend 15 hours in laboratory preparing a project of a simple system in Rational Rose. Software Engineering 2 should widen student's knowledge of software engineering.

Much has been already done in the area of teaching software engineering. Every year there are dedicated international conferences on this subject e.g. Conference on Software Engineering Education and Training or in conferences on software engineering there are special sessions e.g. in Polish Software Engineering Conference [7], Automated Software Engineering, Software Engineering and Applications. In proceeding from these conferences many interesting ideas, curricula's [5], [10],

¹ This work was supported by the Dean of the Faculty of Electronics and Information Systems, Warsaw University of Technology, under grant number 503/G/1032/2900/000.

experiences can be found. The difficulty of teaching software engineering is that it is a multi-faceted discipline. A software engineer must combine formal knowledge, good judgment and taste, experience, and ability to interact with, and understand the needs of clients, work in a team.

Preparing the subjects for Software Engineering 2 course the directions from Software Engineering Body of Knowledge [4] were considered. It was not possible to use all the directions from SWEBOK and present all remaining aspects of software engineering. The chosen subjects are the following:

- requirements engineering,
- design with reuse,
- precise modelling,
- testing,
- quality improvement,
- software effectiveness,
- software processes.

In SE-2 course are 15 hours of laboratory in blocks of two hours. Seven laboratory subjects, correlated with lectures, were chosen. For each laboratory session detailed instructions were written. These instructions make the laboratory sessions supervised by different lecturers similar and students can work in laboratory very effectively.

A software engineer must now be able to pick up a new tool and an associated process quickly, regardless of his or her vested experience in a previous tool. It implies that the student must gain the skills to be able to switch among tools. The goal of laboratory was also to present professional CASE tools supporting different phases of software process. Rational Suite is available at the Institute of Computer Science and the Institute of Control & Computation Engineering. During SE-2 laboratories following tools are used: Rose, Requisite Pro, PureCoverage, Quantify and Purify. These tools are complicated and it is difficult to use them without training. Rational tutorials for these tools available in English, are rather long and can not be used during laboratory exercises. A set of tutorials for these tools in Polish, native language for students, were prepared. Tutorials and laboratory instructions are stored on the department server and are available to students, lecturers and instructors of SE-2 course. Each student can study the instructions and tutoring materials at home, so the instructor's directions can not surprise her/him.

The SE-2 laboratory was introduced in fall 2004 for seven laboratory groups at the Institute of Computer Science. About fifty students of specialization Engineering of Information Systems were the first users of these materials and instructions. Currently the next group of students (56) just finished SE-2 laboratory based on these instructions.

In Section 1 some information about SWEBOK is given. In next sections the SE-2 laboratory instructions are briefly presented. The results of laboratory are discussed in Section 4. The SE-2 laboratory tutoring materials were prepared by I. Bluemke, A. Derezińska, M. Nowacki, P. Radziszewski in the fall 2004.

1. SWEBOK

The Guide to the Software Engineering Body of Knowledge (SWEBOK) [4] has been developed as a guide to generally accepted software engineering knowledge. The 2001 Trial version 1.0 of the Guide has been world-widely discussed and reviewed. The 2004 edition of SWEBOK Guide has been published on the project's web site [4] in the fall of 2004 and will be available in print in 2005. ISO/IEC JTC1/SC7, the international standards organization for software and systems engineering, is adopting the SWEBOK Guide as ISO/IEC Technical Report 19759. The 2004 Guide will continue the refinement process to meet the needs of the software engineering community. The next product, reviewed and approved by the Computer Society, is expected to appear in 2008.

In the current version of SWEBOK the software engineering knowledge is divided into ten Knowledge Areas (KAs):

1. software requirements,
2. software design,
3. software construction,
4. software testing,
5. software maintenance,
6. software configuration management,
7. software engineering management,
8. software engineering process,
9. software engineering tools and methods, and
10. software's quality.

Each KA consists of hierarchical breakdown of topics, reference topics, related reference materials and a matrix linking the topics to the reference materials.

SWEBOK intended to cover the material consistent with the design point of bachelor's degree plus four years of experience (for USA education system). The guide does not define curricula, but it can assist in their development and improvement [3] as each knowledge area is decomposed into topics and associated with ratings from Bloom's taxonomy [1]. Bloom's taxonomy recognizes six cognitive educational goals: knowledge (K), comprehension (C), application (AP), analysis (AN), evaluation (E), and synthesis (S). SWEBOK includes mappings between general topics within KAs and these categories. Practical application of the taxonomy requires the adaptation for the desired profile of a student. For example, the mapping of topics for three software engineer profiles: a new graduate, a graduate with four years of experience, and an experienced member of a software engineering process group were discussed in [2] for four KAs: software maintenance, software engineering management, software engineering process, and software quality.

Teaching goals in software engineering refer to different abilities [10]: any student should know basic principles in all facets of SE, know existing body of knowledge, apply current techniques methods and tools (as examples), and finally understand effects (pro's & con's) of competing t/m/t's for different contexts. Achieving these goals requires different forms of student activities.

There is still a gap between the material recommended by SWEBOK and even taught during lectures, and the topics practiced during exercises in laboratories. The

SE-2 laboratory relates to the following topics (and sub-topics) of the six KAs of SWEBOK.

- (1) – Software requirements: fundamentals (functional and non-functional requirements), requirements process, requirements elicitation, analysis, specification, validation (reviews), practical consideration (requirements attributes, requirements tracing).
- (2) – Software design: software structure and architecture (design patterns), software design quality analysis and evaluation, software design notations, software design strategies and methods
- (3) – Software construction: practical considerations (construction languages, coding construction quality).
- (4) – Software testing: objectives of testing (functional testing), test techniques (code-based techniques, selecting and combining techniques, test-related measures – coverage/thoroughness measures), test process (test documentation, test activities).
- (9) – Software engineering tools and methods: software requirements tools, design tools, construction tools, testing tools (test execution frameworks, performance analysis tools), software engineering process tools, heuristic methods (object-oriented methods), formal methods (specification languages and notations)
- (10) – Software quality: software quality management process (verification and validation, technical reviews).

In Table 1 the comparison of taxonomy levels recommended in SWEBOK and anticipated in SE-2 are presented. As an example the topics from KA (1) – software requirements are shown. Due to time limits some levels are planned to be lower than those in SWEBOK. Selected topics can be further exercised during non-obligatory courses, which can be chosen by the students. For other KS subjects, especially testing and software engineering tools and methods, the levels anticipated in SE-2 are the same as in SWEBOK.

Table 1. Taxonomy levels for KA software requirements

Selected topics	Taxonomy level (Bloom)	
	SWEBOK	SE-2
1. Software requirements fundamentals		
Definition of software requirement	C	C
Functional and non-functional requirements	C	AP
2. Requirements process		
Process actors	C	C
3. Requirements elicitation		
Requirements sources	AP	C
Elicitation techniques	AP	C
4. Requirements analysis		
Requirements classification	AP	AP
Architectural design and requirements allocation	AN	AP
5. Requirements specification		

	System definition document	C	C
	Software requirements specification	AP	C
6.	Requirements validation		
	Requirements reviews	AP	AP
7.	Practical consideration		
	Requirements attributes	C	AP
	Requirements tracing	AP	C

2. Structure of Internet Materials for SE-2

The SE-2 laboratory materials are written in html and stored on a department server. Only students and lectures of SE-2 course have access to the material. The structure of SE-2 laboratory materials (Figure 1) is following:

- Introduction containing SE-2 laboratory goals,
- Tutoring materials,
- Laboratory directions,
- Vocabulary,
- Bibliography

At every screen links to the “parent” section and the main index, to next and previous sections (Figure 2) are available. In each laboratory instruction there are links to the appropriate tutoring material and positions in vocabulary.

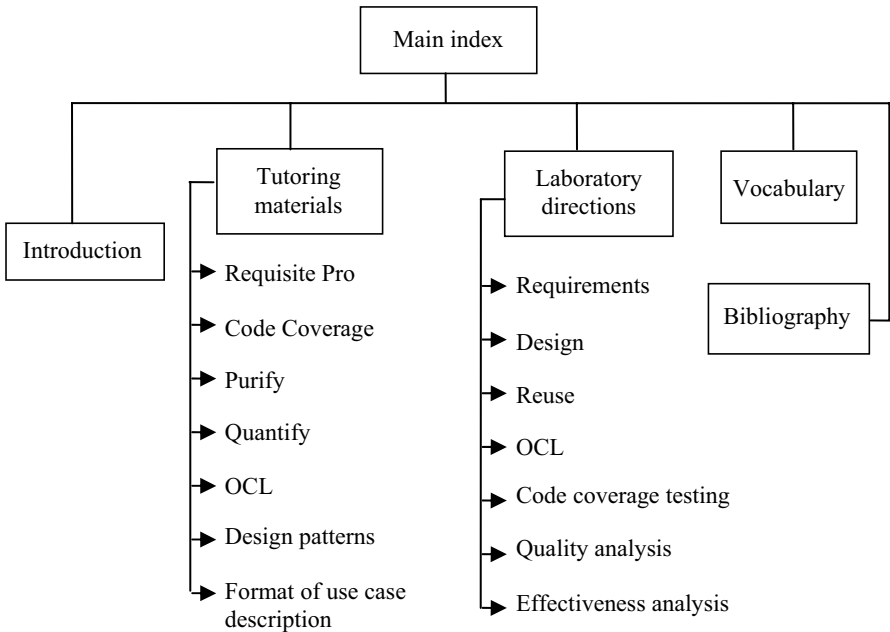


Figure 1. Structure of SE-2 Internet materials

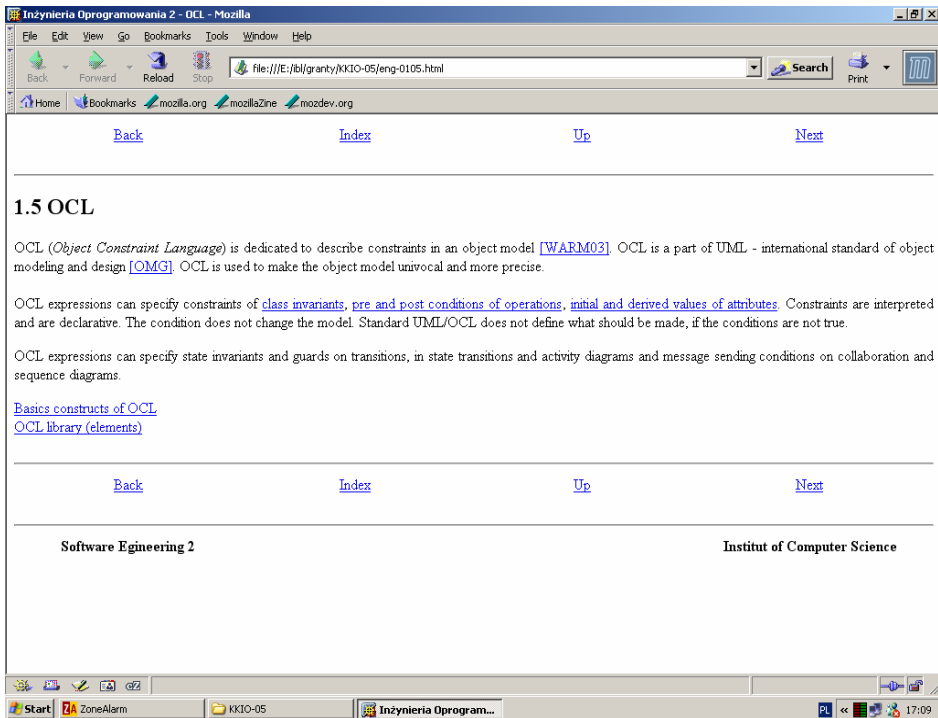


Figure 2. Screen from SE-2 Internet materials

3. Laboratory Subjects

The main goal of Software Engineering 2 laboratory was to refine student's knowledge and experience. As ten SWEBOK knowledge areas, presented in Section 1, can not be included into only fifteen laboratory hours, some of them have to be skipped. Seven laboratory exercises were developed. The subjects of these exercises are following:

- Requirements engineering
- Software design
- Reuse
- Precise modelling with OCL
- Code coverage testing
- Quality improvement
- Improving effectiveness

Subjects of exercises one, and three to seven, are completely new to students. In all laboratory exercises tools from Rational Suite: Requisite Pro, Rose, PureCoverage, SoDA, Purify, Quantify are used. Students on the sixth semester are familiar only with Rose and Soda, because these tools were used on software engineering laboratory in the former semester. Rational Suite provides tutorials (in English) for most tools. Each

of these tutorials needs at most two hours, so they are too long to be used during laboratories. During SE-2 lectures there is no time to present and discuss CASE tools in details. Problems with a tool usage could disturb the student in the development of laboratory subject. To avoid this problem short tutorials in Polish (native language for students), were prepared. All these tutorials are stored on the department server and are available to SE-2 students all semester. However OCL and design with reuse is presented on SE-2 lectures, some tutoring materials to these subjects are also available.

The ability to work in a team is very important for a software engineer and should be trained as well. Interesting proposal how to teach team work can be found for example in [6]. However developing the SE-2 laboratory we were aware of the significance of the teamwork, we decided to incorporate such training in this laboratory only in a very limited extent due to the time constraints (only 15 hours of laboratory). In SE-2 laboratory an instructor – lecturer has a group of seven to eight students. A group of students is preparing requirements for a common subject, divided into individual subtasks. The members of the group cooperate with each other coordinating the relations between the subtasks. To signal some teamwork problems, the specification of requirements made on the first laboratory, is reviewed by another student. Further, on the second and the third laboratory, the reviewer is designing the system, using the reviewed by her/him specification. The instructors were surprised to notice, that students were able to detect in their reviews errors, similar to errors they made themselves in their own requirements specification.

Usually five instructors are having SE-2 laboratory groups. If there are many laboratory instructors, students often complain, that the instructor's requirements towards students, are not the same. To solve this problem detailed directions for each laboratory exercise are given and stored on the department server too. Each of laboratory instruction consists of four parts:

- Aim of exercise,
- Preconditions,
- Detailed directions,
- Exercise results.

In each laboratory instruction there are links to tutoring materials, bibliography and description of difficult ideas in the vocabulary part. In Section 3.1 translation of a part of the instruction to exercise five – testing with code coverage is given.

3.1. Directions to the Exercise with Code Coverage Testing

Below an extract from a translation of the fifth SE-2 exercise instruction concerning code coverage testing is presented. *Italic* font is used to denote Pure Coverage buttons, options and underline shows links to other parts of SE-2 internet materials, dots (...) indicate the omitted parts of the instruction:

The goal of exercise is the quality assessment of test cases. The set of test cases is chosen according to a criterion of structural testing (white box) – coverage of the program code [8], [10]. The program will be analyzed with Pure Coverage tool.

Preconditions

A program in C/C++ is necessary (a program from other courses ex. Algorithms and Data Structures, Compiling Techniques can be taken). The program should have a non-

trivial control structure (with many branches). (...) An initial set of functional tests (black box) comprising test cases (program input parameters, input files, output files, sequences read from the keyboard, etc.) should be prepared.

Tasks

1. Run *PureCoverage* and set its parameters. (...)
2. Run first test case from the initial set.
 - Select *Files* → *Run* and give the location of the executable task in *Run Program* dialog.
 - Define paths for the instrumented code. (...)
 - In *Run* dialog define a location of the working directory for result files and command-line arguments, if required. (...)
 - After the execution of *Run* command, a modified executable task (the so-called instrumented) will be created in Cache directory. Then this task will be executed.
3. Analyze results of program run
 - Compare the number of calls, number of functions missed and hits, lines missed and hit, and the percentage of coverage for particular modules, files and functions (windows *Coverage Browser* and *Function List*). Total results for a run are showed in window *Run Summary*.
 - In widow *Annotated Source* check, which lines of code where covered during the run (denoted with different colors). (...)
4. Run the program for consecutive tests from the initial set of functional tests. The program can be executed with different arguments, using *File* → *Run* and giving new arguments in *Command-line arguments* box. (...)
5. Compare results for many program runs and the merged results for the set of tests. The results for all executed tests are summarized in *AutoMerge* description. Results of the previously executed runs or for the sum of runs (all runs – *AutoMerge* or a selected subset – *Merge*) can be displayed after double-click to a given line in the browser.
6. Design new test cases to increase the merged coverage, if possible.
7. Select an “optimal” set of tests – it is a minimal set of tests, for which the merged coverage is the maximal.

Results

Final results of the laboratory comprise the tested program (source code and the executable task), set of test cases and remarks. In remarks include:

- Brief description of the program – functionality, author and origin (e.g. URL address), users guide.
- Specification of test cases. For each test case define identifier, functional requirements checked by this test, input data (input values, sequence of pressed buttons, input files, etc.), expected results.
- Results of function coverage and source code coverage, collected for each test case.
- Merged coverage results for initial set of tests and the set of all tests (see. p. 5)
- Description of the following three sets of tests: initial, additional tests designed to maximize the coverage (if exists) and “optimal” (see p.7).
- Comments about the results and in particular about the parts of the code, which were not covered.

4. Results of Exercises

In Section 4.1 some results of three laboratory subjects: coverage testing, quality improvement, and improving effectiveness (exercises 5, 6, 7 mentioned in Section 3, are discussed. Further, the evaluation of the students results in SE-2 laboratory is given.

4.1. Results of Exercises 5, 6 and 7

After laboratory exercise students were obliged to send final remarks by email. The contents and structure of these remarks is determined in laboratory instruction (example in Section 3.1). For this instruction many interesting, and surprising to students, results were obtained. Several students were testing, during the fifth exercise, programs written as a project on Compiling Techniques course, because the same students are attending this course as well. Some students were testing a program checking, if a string of characters is generated by the given regular expression. They started with initial set of tests covering the program code in about 85%. Then, they were adding new tests to cover lines not executed so far. For eleven test cases the total code coverage was 99,13% (two lines not covered). From this set they selected two test cases enabling almost the same code coverage. They were surprised with such results.

The most of uncovered code related to the exception handling, not used methods (e.g. overloaded constructors, get or set functions), parts of code used in previous versions of the program. Creating tests triggering some exception routines was difficult. A lesson learned by the students from the analysis of the uncovered code was, that the diagnostic code (exceptions, evaluation of assertions and other fault-tolerant mechanisms) should be validated via code inspection, if special testing is not economically justified. Another observation was, that the program code refactoring is necessary after the essential modification. Removing not needed parts of code decreases further costs of testing and maintenance.

In next exercise students were using Purify to find memory leaks in programs. Again they were astonished detecting memory errors in programs they thought are fault free. The errors were caused mostly by non-careful memory management, for example exceeding array bounds during a read/write operation, freeing an invalid memory in delete operation, reading from a freed memory. Many errors could have been easily located in appropriate destructors. The remaining errors could be found using the references to the source code provided by the Purify tool. After this exercise the students planed to avoid such errors and use program analyzers more often, or to implement programs using less error-prone languages like Java or C#.

During the last exercise they were observing efficiency of a program in Quantify and improving it by removing its bottlenecks. Some students observed that a thread spends the most of the execution time in `printf` function (e.g. 95%). After a simple reorganization of the program (removing of `printf` from the inner function) the entire execution time of the thread function and its descendents was shorter (27 times shorter for the one-threaded program version and 22 times shorter for the multi-threaded one). This experience convinced students about the high cost of input-output operations and benefits of the careful design. If data should be transferred more often some additional buffering mechanisms (cache) can be considered.

After successful optimization a new bottleneck becomes often a system function (e.g. `ExitThread` – the function closing a thread). This showed that a simple

optimization is no more applicable, unless the reengineering of the general program structure is performed. The basic optimization process for the multi-threaded programs was typically finished, when the new bottlenecks referred to synchronization functions, like `WaitForSingleObjectEx`, `ReleaseMutex` (Figure 3).

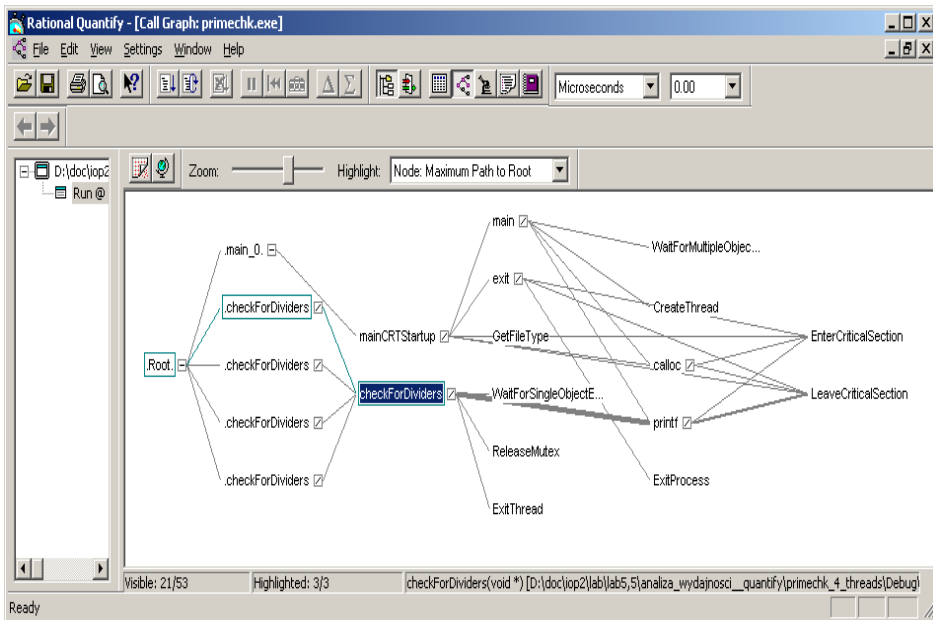


Figure 3. Efficiency analysis – call graph of functions with the critical path

The implementations of the same tasks in one and multi-threaded versions were compared. Students typically observed longer execution time (about few percent) for the multi-threaded solutions. This result was obtained due to overhead required for the thread management and because all threads of the programs were run on the same computer. The observation of states of the threads helped in recognizing errors in cooperation between threads, although no erroneously outputs were detected during the runs of functional tests. In other cases, when the algorithm could be efficiently distributed, the execution time of the multi-threaded version was shorter (even to 40%).

4.2. Evaluation of Exercises

Evaluation of all exercises was based on:

1. An activity of a student (including the way of the approaching to the solutions) and presented partial results during exercises in laboratory.
2. Realized tasks (specifications, models, programs) and final remarks received via email.
3. Critical discussion about received results.
4. A review of results by a colleague (only for first exercise).

The received results contributed to the score of an exercise.

Comparing the scores of different subjects it was stated that the highest notes were obtained for exercises 5, 6, 7 (see Section 3). The subjects of these exercises required many manipulations with the new tools (what could be done easily using instructions and tutorials) but less creative work than subjects of exercises 1–4. Moreover the students were more capable and willing to cope with any problems related directly to the running programs, than to the problems concerning requirements and models. The analyzed programs were not very complicated, because the main goal was learning the new methodology. It was observed, that the preconditions specifying the program complexity, should be more precise in laboratory instructions.

Apart from mentioned in Section 1 mapping to the Bloom taxonomy SWEBOK points out to different skills desired in several topics of the KAs. For example, in software requirements engineering, good communications and negotiations skills are helpful. Decision and matching abilities could be useful while developing program model with reuse. Objective evaluation of students' skills is very difficult and effort-consuming. It was possible because a tutor has a relative small number of students (seven to eight). The anticipated skills were developed in cases, when the starting level of knowledge was satisfactory. The general problem was the requirement of knowledge, which was taught on other courses. For example, OCL constraints could not be applied without a proper construction of UML models presented on the former semester.

5. Conclusion

The SE-2 laboratory comprises phases of software process like requirements engineering, system design with reuse and design patterns, precise modelling with OCL, code coverage testing, and improving the quality and effectiveness of an application. To our best knowledge most of these subjects are not widely used in curricula of obligatory software engineering courses. The SE-2 laboratory relates to six (from ten) subjects of the knowledge areas of SWEBOK. The taxonomy levels recommended in SWEBOK and anticipated in SE-2 for three exercises are the same. Due to the time limits the levels for remaining SE-2 exercises are planned to be lower, then those in SWEBOK. Despite the best efforts of computer science educators, students often do not acquire the desired skills that they need in the future work.

Students believe that once a program runs on sample data, it is correct; most programming errors are reported by the compiler; and debugging is the best approach when a program misbehaves. We have shown, that in a short time, they can gather experiences broadening their understanding of engineering a program.

Our intents were to use industrially proven tools in SE-2 laboratory. By providing the tutoring materials student could take advantage of the information they provide without being exposed to the hassles of learning to use the tools.

The SE-2 laboratory was introduced in fall 2004. The experiences are still too limited to be able to draw general conclusions. So far it was observed that students were able to do much more than in the previous semesters without Internet instructions. The demands of different instructors, placed to students, were similar. Instructors devoted more attention to the design assessment and refinement.

References

- [1] Bloom B.S. ed., Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain, David MC Kay Company New York, (1956)
- [2] Bourque P., Buglione L., Abran A. & April A., Bloom's Taxonomy Levels for Three Software Engineer Profiles, In Post-Conf. Proc. of Workshop on the Expected Levels of Understanding of SWEBOK Topics, in STEP 2003, September 19-21, (2003), Amsterdam (Netherlands), 123-129
- [3] Frailey D.J., Mason J., Using SWEBOK for education programs in industry and academia. In *Proceedings of 15th Conference on Software Engineering Education and Training, CSEE&T*, Feb. 25-27, (2002), 6-10
- [4] Guide to the Software Engineering Body of Knowledge, (2004), www.swebok.org
- [5] Jazayeri M., The education of a Software Engineer. In *Proceedings of the 19th Inter. Conf. on Automated Software Engineering, ASE'04*, (2004)
- [6] Nawrocki J.R., Towards Educating Leaders of Software Teams: A New Software Engineering Programme at PUT, in P. Klint, J.R. Nawrocki (eds.), *Software Engineering Education Symposium SEES'98 Conference Proceedings*, Scientific Publishers OWN, Poznan, (1998), 149-157
- [7] Nawrocki J.R., Walter B.(eds.), In polish: Wybrane problemy inżynierii oprogramowania. Nakom, (2002), part 5
- [8] Patton R., Software testing. Pearson, (2002). In polish: Testowanie oprogramowania. *Mikom*, (2002)
- [9] Pressman R.S., A software engineering a practitioner's approach. Mc GrawHill, (2001), In polish: Praktyczne podejście do inżynierii oprogramowania. *WNT*, (2004)
- [10] Rombach D., Teaching how to engineer software. In *Proceedings of 16th Conference on Software Engineering Education and Training, CSEE&T*, Mar. 20-22, (2003)
- [11] Unified Modelling Language Specification, www.omg.org/uml
- [12] Warmer J., Kleppe A., The Object Constraint Language Precise Modeling with UML. *Addison Wesley*, (1999), in polish: OCL precyzyjne modelowanie w UML. *WNT* (2003)

Configuration Management of Massively Scalable Systems

Marcin JARZAB, Jacek KOSINSKI and Krzysztof ZIELINSKI

*Institute of Computer Science,
University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
e-mails: {mj, jgk, kz}@agh.edu.pl
<http://www.ics.agh.edu.pl>*

Abstract. The number of computational elements in massively scalable systems can easily reach several hundred of processors. The exploitation of such systems creates new challenges. The paper presents the contemporary technology of massively scalable computer systems configuration management. This problem has been put in context of modern hardware resources virtualization techniques. The paper refers to solutions provided in this area by SUN Microsystems to show real existing solutions which are used and practically tested by the authors of this paper.

Introduction

Modern massively scalable computer systems usually consist of many replicated computer nodes interconnected with high speed communication subsystems. Very often they take the form of homogeneous clusters of computers or hierarchical multi-tier systems where each tier represents a collection of computers. The architecture of such systems e.g. CRM, ERP or Internet portals is driven by requirements of large organizations' data centers, offering access to wide spectrum of services. From the software architecture's point of view they often follow the Service Oriented Architecture (SOA) [1] paradigm and exploit Java 2 Enterprise Edition (J2EE) [2] architecture extended with J2EE Connector Architecture (JCA) [3] to provide access to legacy systems. The activity of such systems requires efficient mechanisms for on-demand resource allocation and sharing, isolation of computation of independent groups of users and accounting of resource consumption. Isolation means that computations should be performed not only in separate security domains but should be separated on the level that is offered when they are performed in different computers interconnected by computer networks.

The number of computational elements in massively scalable systems can easily reach several hundred of processors. The design and exploitation of such systems creates new challenges, addressed by research in the area of Grid [4] or utility computing [5]. Utility computing involves a fusion of technologies, business models and infrastructure management techniques. Specifically, utility computing accomplishes the goal of greater IT resource flexibility and efficiency. It is achieved through intelligently matching IT resources to meet business demands.

The goal of this paper is to present the contemporary technology of massively scalable computer systems configuration management as a new and very important part of software engineering. This problem has been put in context of modern hardware resource virtualization techniques and high density multicomputer systems construction. It is necessary to show the complexity and functionality of configuration management systems. The paper refers to solutions provided in this area by SUN Microsystems to show real, existing solutions which are used and practically tested by the authors of this paper. Despite addressing concrete systems, the presented considerations are far more general in nature and they present new trends in next-generation computer system construction.

The structure of the paper is as follows. In Section 1 the massively scalable architecture is very briefly described. Hardware and systems components are characterized together with virtualization layers. Subsequently, in Section 2, configuration management requirements of large computer systems is discussed and the main problems in this area are identified. Concrete examples of such tools are presented in Section 3. Finally, Section 4 contains practical examples of large system configuration management. The paper ends with conclusions.

1. Massively Scalable System Architecture

This section details the deployment patterns and implementation strategies used to achieve massive scalability. This influences the hardware configuration of computer systems and the virtualization techniques used for efficient resource management.

1.1. Architecture of Massive Scalable Systems – Overview

Distribution techniques allow the system to be scaled by adding additional commodity hardware as needed, taking an incremental approach to scaling rather than requiring to retrofit the hardware platform with more physical resources such as processors, memory or disks etc. Scaling the system is a matter of configuring the deployment environment as necessary rather than rebuilding or re-customizing the applications themselves every time the load requirements or use cases change.

System performance and capacity overhead/degradation as a result of employing these distribution patterns should be predictable enough to be suitable for planning purposes. Documented best practices [6] and deployment strategies take the form of design patterns and implementation recommendations that address scalability issues across a number of architectural areas common to applications deployment (see Figure 1), namely:

- Scaling of logic: Serving very large numbers of concurrent clients within an adequate response time,
- Scaling of data: Integrating many back-end systems without compromising performance and scalability,
- Scaling of events: Managing large numbers of events, e.g. billing, performance metrics, faults without compromise.

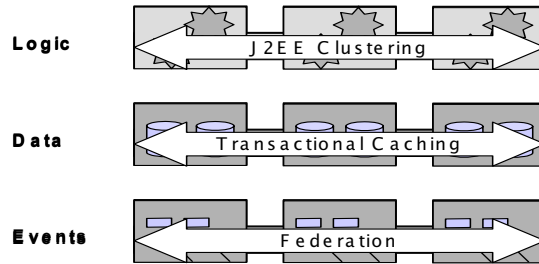


Figure 1. Scalability patterns

The deployment architecture of such systems consists of a number of tiers, each of them responsible for a specific logical function within the overall solution. Each tier is a logical partition of the separation of concerns of the system and is assigned a unique responsibility. The whole system is represented as a stack of tiers, logically separated from one another. Each tier is loosely coupled with the adjacent tier and can be scaled independently of the others.

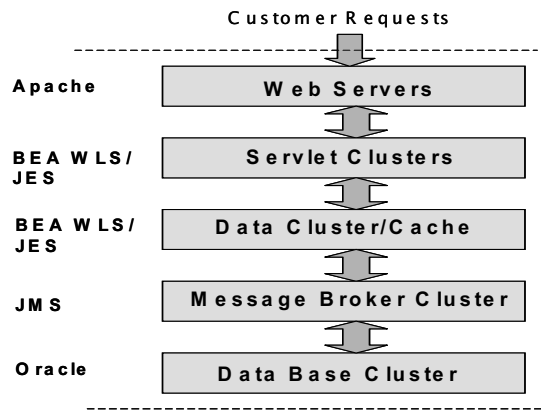


Figure 2. Typical multi-tiered system

An example of such a system is shown in Figure 2. It consists of the following tiers:

- Web Servers – accepts customer HTTP requests and passes them to the servlet cluster layer,
- Servlet Cluster – responds to HTTP requests from the web servers with the appropriate customer information from the underlying data cluster,
- Data Cluster – supports a unified view of all customers and their related order status,
- Message Broker Cluster – provides the infrastructure for managing the propagation of events from the back-end operational systems to the data cluster,
- Service Activation – the back-end systems that own the customer information.

1.2. Deployment Patterns

Deployment patterns and implementation strategies are designed to promote performance and scalability in a typical applications integrated environment. A deployment pattern defines a recurring solution to a problem in a particular context. A context is the environment, surroundings, situation, or interrelated conditions within which something exists. A problem is an unsettled question, something that needs to be investigated and solved. A problem can be specified by a set of causes and effects. Typically, the problem is constrained by the context in which it occurs. Finally, the solution refers to the answer to the problem in a context that helps resolve the issues.

Detailed analysis of a massively scalable system leads to the conclusion that most commonly used deployment patterns through every tier of the system depicted in Figure 2 are as follows:

- Application server pooling and clustering,
- Transactional Data Caching,
- Cache routing.

The clustering technique assumes that execution components are replicated and deployed over a larger number of computing elements and there exists a system element that is responsible for load distribution among them. This means that a large number of execution components must be effectively managed in each tier of the system. The Data Caching pattern assures that most of read data are available locally and remote accesses are eliminated through a Cache Routing pattern [6].

1.3. Blade – Architecture

From the already discussed requirements of massively scalable systems it is evident that the most suitable hardware architecture is a cluster of computers that may be used in each tier. Unfortunately, a typical cluster built of independent computers interconnected with a high-speed communication network e.g. gigabit Ethernet, does not support the following:

- Higher server density to reduce the amount of floor and rack space required,
- Increased server utilization by just-in-time provisioning of servers to services as customer demands fluctuate,
- Shared infrastructure that reduces per-server energy consumption and cooling requirements while simultaneously increasing availability,
- Reducing administration costs by simplifying the physical plant, and also by managing a pool of servers as a single resource, with provisioning and re-provisioning simplified through state-of-the-art management software.

Searching for the solution, the largest hardware vendors such as Intel, IBM, HP and SUN have invented the so-called blade architecture. This architecture assumes that a single board computers, called blades, are plugged-in into a rack-mount “intelligent” shelf. The shelf may host typically up to 16 or 24 blades and is equipped with a communications subsystem and control logic that may support system redundancy. In our

study we have used the SunFire Blade Platform architecture. The Sun Fire B1600 Intelligent Shelf is depicted in Figure 3.



Figure 3. Sun Fire B1600 Intelligent Shelf

1.4. Virtualization

The most effective technique for complex hardware resource management is their virtualization. The best-known example of this technique application is an operating system which virtualizes computer resources providing standard APIs. Clusters of computers as well as large Symmetric Multi-Processing (SMP) computers require new levels of virtualization for their effective utilization.

In the case of computer clusters, virtual farms can be easily organized. Each virtual farm consists of a collection of computers interconnected with VPNs. This way each farm is isolated from the others and may be handled as an independent cluster.

For SMP computers, the container technology [7] or logical/virtual partitions [8] are offered by leading operating system producers. These technologies support virtual servers, allowing us to partition a single physical server into more than one protected operating environment, with applications running in each virtual server essentially isolated from all others. One of the most scalable virtual server technologies is the Sun Solaris Containers technology implemented by Solaris 10 OS. This system can run up 8192 containers on a single server.

Both virtualization levels introduce a large number of logical elements that must be efficiently configured and managed. This fact places configuration management at the forefront of contemporary massive scalable system requirements.

2. Configuration Management of Large Computer System Requirements

Configuration management of applications running over virtualized resources is a sophisticated process, due to the large number of logical elements which have to be managed and the complexity of contemporary application installation – the deployment process is dependent on many parameters which must be customized for each computational element of the system. This makes configuration management time-consuming and error-prone, creating the risk of a system breakdown.

The list of typical requirements related to configuration management of large computer systems is summarized in Table 1. These requirements put stress on automation of the deployment process over large numbers of computational nodes, deployment process simulation, real-time configuration file generation, configuration comparison and checking. This list is extended by requirements related to version control logging and reporting.

Table 1. Typical requirements of configuration management tools

Requirement	Description
Automated Deployment	End-to-end automation of the deployment process, including distribution, configuration, and startup of packaged and custom applications.
Differential Deployment	Enabling delta-only distribution of large content directories thereby speeding up deployments significantly and optimizing incremental directory updates.
Deployment Simulation	Complete simulation of deployments to ensure that key requirements for success are in place before deployment.
Dynamic Configuration	Real-time generation of application configuration for the target environment provides flexibility during deployment.
Dependency Management	Ability to encode application dependency information which is checked during Deployment Simulation to prevent errors that cause downtime and to leverages best practices across entire operations team.
Application Comparison	Ability to track application configuration drifts, and pinpoint unauthorized changes to servers reduces the time required for root-cause analysis of application errors.
Version Control	Central repository that tracks all deployment and configuration data for reference, reconstruction, and to automate rollback to previous states.
Logging and Reporting	Detailed logs of every action taken by the system across all applications and managed servers provides complete audit history of every change made to every server.

Configuration management aspects discussed thus far concern the initial setup of the application. Nevertheless, recent trends also concern the application management process during runtime. The initial startup parameters may be changed due to existing load or end-user needs. The most popular standard solution utilizes JMX [9] technology to support this functionality. Examples of such systems include the Sun Java System Application Server 8.1 [10], JBoss [11] and the BEA Weblogic Application Server (WLS) [12].

3. SUN's Configuration Management Tools Overview

This section describes the N1 integrated environment built by SUN Microsystems for provisioning services in large computer systems. It offers the virtualization techniques described before and a scalable deployment tool for application configuration management. N1 does this by creating a virtual system out of the underlying computer network and storage resources.

The N1 Provisioning Server [13] software allows the complete design configuration, deployment, and management of multiple secure, independent, logical server farms. It therefore addresses the first level of virtualization investigated in Section 1. This task can be managed using the Control Center, the main component of N1 Provisioning Server software. Every N1 resource is located in an I-Fabric (Infrastructure Fabric) which combines storage, networking and computing resources

into a contiguous infrastructure that users can deploy and manage to meet changing requirements. This fabric enables management, deployment, and redeployment of logical server farms and is made up of three functional areas:

- Control plane – The control plane consists of the N1 Provisioning Server software and the associated server hardware on which the software is deployed,
- Fabric layer – The fabric layer contains the networking infrastructure and switching fabric. This layer enables the software in the control plane to dynamically create, manage and change the networking and security boundaries of logical server farms. These limits are being realized through the creation of dynamic VLANs,
- Resource layer – The resource layer consists of infrastructure resources such as blade servers, load balancers and SSL accelerators.

Solaris 10 Containers [16] split a single machine into many different operating system instances called zones, with defined resource consumption policies. By default, there is always a global zone from which, if needed, system administrator can create other zones. Each zone has its own IP address, booting capabilities, resource control and accounting. A Zone provides a virtual mapping from software services other to platform resources, and allows application components to be isolated from each other even though they share a single Solaris OS instance. It establishes boundaries for resource consumption and provides isolation from other zones on the same system. The boundaries can be changed dynamically to adapt to changing processing requirements of the applications running in the zone.

These two virtualized layers can be effectively managed by the N1 Provisioning System and the N1 Grid Console.

The *N1 Provisioning System* [14] is a software platform that automates the deployment, configuration, and analysis of distributed applications. It enables organizations to manage applications as distinct units, rather than as large sets of installation files, thus provides the following benefits:

- Centralized control over deployments and configuration,
- Greater coordination among system administrators,
- Increased productivity through automation.

The N1 Provisioning System provides an object model to define a set of Components and Plans for system configuration, service provisioning, and application deployment automation. The N1 provisioning system includes a component library with templates for the most common components in Internet data centers. Operators can capture all the relevant configuration data in a “Gold Server” a.k.a. reference server so that its configuration can be replicated to other servers in a Grid.

It is useful to understand the basic elements of the N1 Provisioning System depicted in Figure 4:

- N1 Grid Service Provisioning System Master Server (MS): this is a central application that maintains user records, server history and state records, modeling definitions, and other global data relating to the application deployment process.

- N1 Grid Service Provisioning System Remote Agent (RA): this is a light-weight application that resides on each server managed by an N1 Grid Service Provisioning System installation. The Remote Agent accesses the local system environment on the server.
- N1 Grid Service Provisioning System Local Distributor (LD): this application acts as a caching proxy for resources to be deployed on servers, and as a forwarding gateway for commands to be executed by an RA.

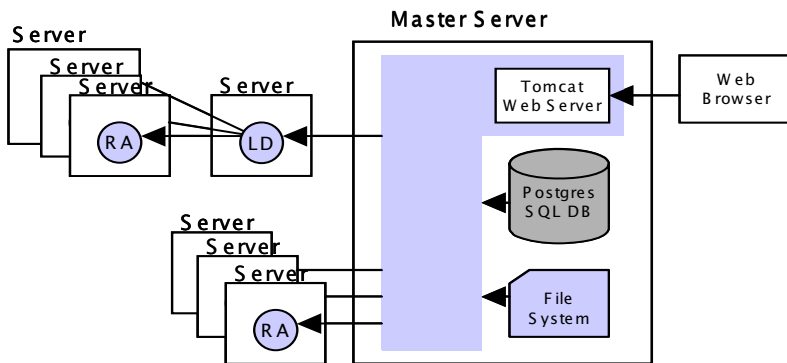


Figure 4. N1 Provisioning System architecture

The *N1 Grid Console* [15] is a browser-based manager for Solaris 9 Containers. It allows administrators to easily partition a single instance of a Solaris 9 OS into multiple containers that can run applications. It must be emphasized that there is a difference between containers in Solaris 9 and Solaris 10. In the latter case, the given container is mapped to a zone with assigned resources and Solaris 9 can support many containers in a single instance of the OS. Solaris 10 Containers can be managed by the N1 Provisioning System.

4. Case Study

This section presents a case study of configuring a large farm of WebLogic Servers (WLS) (see Figure 5). The Provisioning System (PS) Master Server (MS) has been installed on node 0 under Solaris 9 together with a WWW server for load-balancing purposes (Apache 2.0.53 with the WebLogic plug-in). Nodes 1 to 30 run PS Remote Agents and several WLS run in separate Solaris 10 zones. The Nodes Repository stores information about nodes which are present in the farm. This information is used by the MS to check if new nodes are added to farm and eventually add those nodes to the PS management list. Information in the Nodes Repository is updated by the Node Discovery agent started in the Global Zone of each new node added to the farm.

4.1. Preparing for the Provisioning

The preparation of the system being studied concerns mainly process setup and activation of the test farm with the N1 Provisioning Server tool. In this phase stress is placed to a large extent on the automation of subsequent deployment. We have decided

to install a farm containing 31 B100s nodes as 30 nodes under Solaris 10 and a single node under Solaris 9. The default configurations of the operating system were customized to perform functions of the N1 Grid Provisioning System Server (PS) Master Server (MS) (Figure 6) on the node operated under Solaris 9, since the exiting version 5.0 is not available for Solaris 10. Other nodes run the N1 Grid Provisioning System Remote Agent (RA).

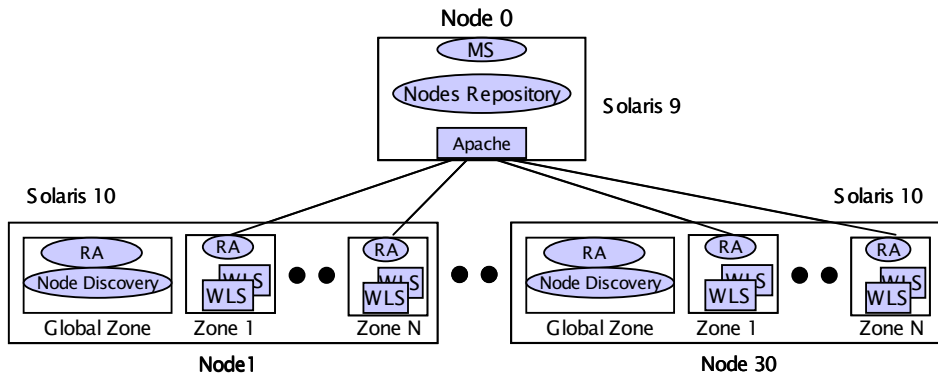


Figure 5. Testing farm configuration

More specifically, the test farm installation was performed in the following steps:

1. Modification of boot scripts of the Solaris system so that this software is run automatically following system startup.
2. Modification of N1 Grid Provisioning System Remote Agent software boot scripts in order to make it work independently of the IP address of the host it is installed on. This proved necessary because this software has to be attached to the image of the operating system and this image cannot have IP addresses bound into its configuration statically. The reason is that the Provisioning Server assigns addresses to nodes during farm creation. Software that checks for the launching of the MS RA component is also installed on the node. It sends information about RA status and parameters periodically to the repository for the given farm. Data collected in the given repository serves for generating a configuration of available nodes in the N1 Grid Provisioning System.
3. Generating of images of:
 - Solaris 10 system with RA software.
 - Solaris 9 system with N1 Grid Provisioning system software.
4. Installation of tools and software:
 - DNS (Berkley Internet Name Domain v. 9.3.1) server used as a method of dynamically configuring IP addressing for newly created zones.
 - Apache Web Server with configured WLS plug-in for clustering purposes. This step consists of copying the already present installation from the “Gold Server” to the target node and adjusting parameters in *httpd.conf* (Table 2) for WLS cluster load balancing.

Table 2. Sample configuration of the Apache plug-in for WLS clustering. Lines 1-3 depict nodes and port numbers at which WLS cluster instances run and *Location medical-rec* is a context (Web application) for which HTTP requests must be load-balanced between WLS cluster nodes.

```

1. <IfModule mod_weblogic.c>
2.     WebLogicCluster wlsnode1:7001, wlsnode2:7001, wlsnode3:7001
3. </IfModule>
4. <Location /medical-rec>
5.     SetHandler weblogic-handler
6. </Location>

```

- WLS on nodes using silent installation procedure with a configuration file defining one administration server. The Weblogic administration server is used by the PS to create new copies of server instances (a.k.a. managed servers). Depending on the defined strategy each instance can run one or several J2EE applications. Furthermore, at least one instance of WLS must act as a “Gold Server” storing previously-installed applications which in turn can be replicated to other instances of WLS clusters or single instances. Unfortunately, mechanisms for managing the WLS infrastructure are not provided by PS by default. Therefore administrators of PS must install special a BEA-Weblogic plug-in which provides functionality for the creation and management (e.g. application deployment, starting/stopping) of both the administration and managed server instances and also automatic cluster creation.
 - Software that adjusts parameters for resource consumption (CPU shares, memory usage, LWPS number) between zones and also for services running in these zones.
5. For each piece of software installed in the previous step, the PS component configuration is generated and tested.

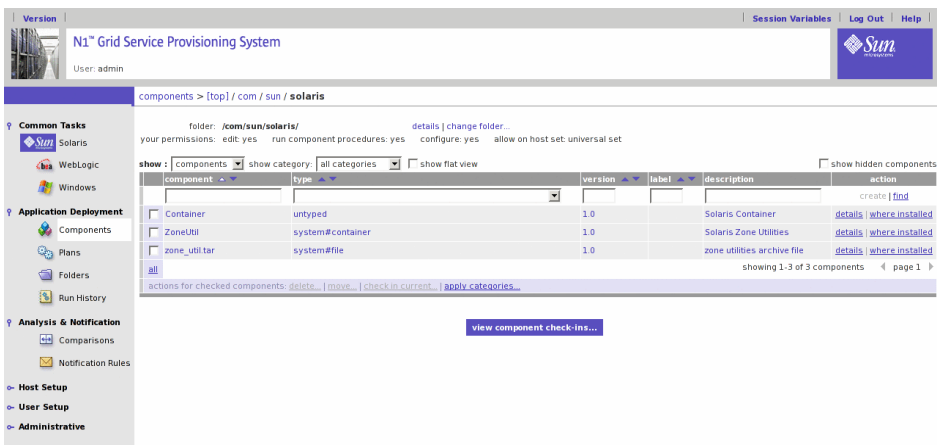


Figure 6. Master Server console with installed components for managing Solaris 10 zones

4.2. Example of Installation Flow

Thanks to the previously-described procedure, a typical deployment scenario is reduced to the following steps:

1. Deployment of the farm using the N1 Provisioning Server which automates the process of creation the physical infrastructure, accelerating the process of farm creation with the usage of predefined operating system images. Each image has already installed N1 PS RA for remote management of software configuration at each node and MS for managing the whole deployment process.
2. Applying the node image with the preinstalled N1 Provisioning System software. The process of the initialization of the PS configuration occurs automatically thanks to applying the node-discovery mechanism. This is done by adding Solaris 10 nodes (from the Nodes Repository) to the configuration of the PS MS host group “global zones”.
3. Installation under PS tools for creating and managing zones. This requires interaction depicted in Figure 7 between N1 PS MS, the DNS system, and software for configuration management (scripts) of resource for zones.
4. Creation and configuration of a suitable number of zones. This process consists of creating the new zones in the following stages:
 - Uploading parameters (IP, Host name) to the DNS server for the new zone.
 - Downloading parameters (IP, Host name) from the DNS server at the moment of initialization of the local zone.
5. Installation of the WLS software on selected zones using silent-installation mode performed by PS. Each installed WLS server has one domain which is managed by the administration server. Following installation, the administration servers configuration (http port, installation directory, admin user and password) at each node must also be specified in the MS, because it is used for creation of new instances (managed servers) of WLS using the MS Web browser console. At least one instance of the managed server must be dedicated to testing purposes (“Gold Server”). The resultant WLS cluster consists of already-installed managed servers. The N1 PS admin can check if the application deployed on the “Gold Server” works faultlessly. If there are no errors, this application can be captured and easily replicated among WLS cluster nodes.
6. Installation of the WWW Apache Server. Applying PS to installation. Automatic configuration of the WLS service plug-in.

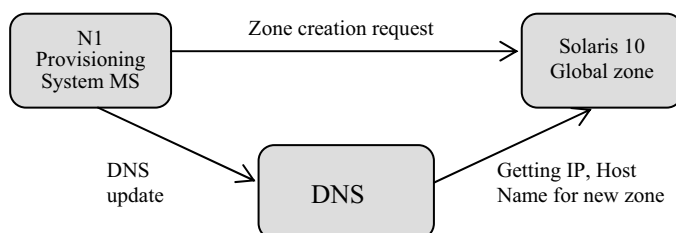


Figure 7. Deployment scenarios interaction for Solaris 10 zones

5. Conclusions

Configuration management of applications running over virtualized resources in large computer systems can be effectively exercised with a new class of software supporting the provisioning process. This process should integrate flexible control of virtualized resources with mechanisms for setting up applications configuration parameters and support for easy replication of exiting installations over large numbers of nodes. The tested SUN tools N1 Provisioning System and N1 Provisioning Server satisfy most of these requirements.

References

- [1] Service Oriented Architecture (SOA) description available at O'Reilly WebServices, <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>
- [2] J2EE 1.4 Platform specification, <http://java.sun.com/j2ee/1.4>
- [3] J2EE Connector Architecture specification, <http://java.sun.com/j2ee/connector/index.jsp>
- [4] Open Grid Service Architecture (OGSA) Specification, <http://www.globus.org/ogsa>
- [5] Utility Computing , Business White Paper SUN Microsystems, March 2004
- [6] Deployment strategies focusing on Massive Scalability, OSS Through Java Initiative, Brian Naughton, April 2003
- [7] Consolidating applications with Solaris containers, SUN Microsystems Technical White Paper, November 2004
- [8] SUN Solaris Container technology vs. IBM Logical Partitions and HP Virtual Partitions, Edison Group's Business Strategy Report
- [9] JMX (Java Management Extension) specification, <http://java.sun.com/products/JavaManagement/>
- [10] SUN Java System Application Server, <http://www.sun.com/software/products/appsrvr/index.xml>
- [11] JBoss application server, <http://www.jboss.org/products/jbossas>
- [12] BEA Weblogic Application Server, <http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/server>
- [13] N1 Provisioning Server Blades Edition, http://www.sun.com/software/products/provisioning_server/index.html
- [14] N1 Provisioning System, http://www.sun.com/software/products/service_provisioning/index.html
- [15] N1 Grid Console, http://www.sun.com/software/products/grid_console/index.xml
- [16] Solaris Containers – What They Are and How to Use Them, SUN BluePrints OnLine, Menno Lageman, May 2005

Interoperability of Monitoring Tools with JINEXT

Włodzimierz FUNIKA and Arkadiusz JANIK
*Institute of Computer Science, AGH,
Al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: funika@uci.agh.edu.pl
phone: (+48 12) 617 44 66, fax: (+48 12) 633 80 54*

Abstract. Monitoring tools are commonly used in modern software engineering. They provide feedback information for both development and for productional phases. The use of monitoring tools is especially important in distributed and grid systems, where different aspects of the environment have to be analyzed/-manipulated by different types of tools, to support the process of program development. However, in order to avoid improper influences of one tool on other ones these tools must cooperate, what is called interoperability. In this paper we present an interoperability model designed and used in the JINEXT extension to OMIS specification, intended to provide interoperability for OMIS-compliant tools. We present a few practical experiments done with JINTOP – the reference implementation of JINEXT.

Keywords. Interoperability, monitoring tools, OMIS, J-OMIS, Java

Introduction

The modern software engineering models usually consist of some phases such as analysis, development, testing etc. The number of phases and their order depends on a type of a model. Independent from a chosen model almost every phase of software engineering is supported by special tools such as compilers, debuggers, editors, profilers etc. In this paper all kinds of tools supporting software creation and maintenance are called *monitoring tools*. The monitoring tools can be provided as an *Integrated Development Environment* (IDE) as well as a set of independent tools from different vendors. Using monitoring tools is necessary to simplify, accelerate as well as to minimize the number of errors and bugs done during the software engineering process because of it's complexity and difficulties. However, using tools which cannot cooperate properly may not accelerate the process but even disrupt it, unless a special coordinating facility, like *monitoring system* provides a kind of *interoperability* support. Interoperability is the ability of two or more software systems to cooperate with each other.

In this paper we present some issues of adding an interoperability-oriented extension to Online Monitoring Interface Specification (OMIS), which provides an interoperability support for monitoring tools. Section 1 gives a short overview of monitoring tools and possible ways of cooperation between them. In Section 2 we focus on the *mediator* interoperability model. Section 3 presents JINEXT, an extension

to the OMIS specification, while in Section 4 we present some practical example of using JINEXT and the results of experiments carried out. In Section 5 we discuss the on-going work on JINEXT and further research.

1. Monitoring Tools and their Interoperability

A typical definition of monitoring tool is as follows [9]:

Monitoring tool – is a run time application which is used to observe and/or manipulate the execution of a software system.

In this paper we use the *monitoring tool* wider, as a tool supporting the process of software development. Monitoring tools can be classified using many different criteria. One possibility is the functionality criterion. According to it monitoring tools can be divided into the following types:

- debugger
- performance analyzer
- load balancer
- flow-visualizer
- controlled deterministic executor
- consistent checkpointing
- editor
- compiler

When considering the operation modes of monitoring tools, the most important object of monitoring and analysis is *time*. Due to this criterion, monitoring tools can be divided into two different types:

- off-line monitoring – the data about the execution of a monitored application is gathered during execution but an analysis of the data is done later, after the application stops. This type of monitoring disables interactions between the user and the monitoring tool.
- on-line monitoring – the monitoring is done during the execution time, allowing the user to interact with monitoring tools all the time. It is possible to change measurements, define new ones and analyze the results at run time.

This criterion does not fit such tools as editors and compilers but fits perfectly into load balancers, debuggers, profilers, flow visualizers etc.

In this paper we address interoperability in the context of monitoring tools, as a kind of *cooperativity* between them on the semantical level, where an interpretation of actions and events provided by each cooperating tool must be guaranteed, e.g. by informing the tool when another tool performs some action, important from the first tool's point of view.

To better understand this problem we present an example interaction between different types of tools. A sequence diagram in Figure 1 illustrates the following scenario:

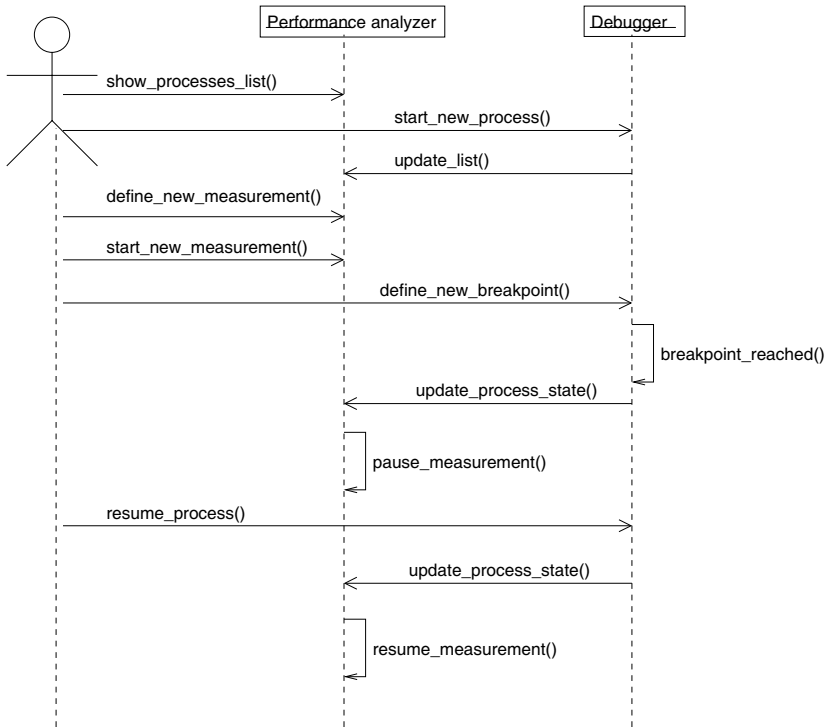


Figure 1. The sequence diagram for the example scenario

1. the performance analyzer displays the list of applications started on the given node;
2. debugger starts a new application;
3. the list of the started applications displayed in the performance analyzer window is updated;
4. the user defines a new measurement for the newly created application and starts it;
5. the user defines a breakpoint in the application;
6. the debugger reaches the breakpoint in the monitored application;
7. the performance analyzer changes the state of the measurement to “paused” to avoid counting the CPU time while the application is not running, the information about the new state of the application is displayed in a sufficiently comprehensive way;
8. the debugger resumes the application;
9. the performance analyzer changes the state of the measurement to “running” and resumes the calculations, the new state of the application is visible on a performance analyzer display;

2. Models of Interoperability

A number of different interoperability models can be found. They vary from solutions basing on interpreting tools as agent systems. LARKS [6] uses specialized layer

(*matchmarker*), responsible for intermediating between the tools. The *hardware metaphore* model can also be used as done in the Eureka Software Factory (ESF) and Purtillo [1]. This solution uses a special language to describe mappings between logical modules (tools) of the system.

In the broadcast message server (BMS) solution a specialized server is used to coordinate communication among different tools. Each tool informs the server about the actions it supports. If a given tool wants to invoke an action provided by another tool, the server is used as the intermediary. If a given tool wants to inform other tools about its events also sends a special announcing message to the server. Every tool should subscribe for events it wants to be notified about.

The encapsulation method combines software components into a single, bigger system. A system object is called *wrapper* and encapsulates individual tools. Owing to encapsulation these tools cannot be accessed directly by clients. The only access to tools' methods is possible from within the body of wrapper routines. By calling a wrapper's method by the user, the wrapper not only executes software object's procedures but also does some integrity steps. Direct access to tools' methods may result in improper integration of objects. On the contrary to the encapsulation method, hardwiring needs the source code of tools be modified. If two monitoring tools need to cooperate, they have to make calls to each other. It means that the programmers have to insert a call to compiler's operation in a source code of the editor's save method. The implicit invocation solution is very similar to hardwiring. The main difference is providing different relationship structures between tools. A given tool can register another tool to be invoked by its events. Instead of the editor calling the compiler, the compiler can register with the editor, to be notified when the editor announces a *file saved* event.

The interoperability model proposed in JINEXT is extending a commonly known *mediator* model [3]. A scheme of this model is presented in Figure 2. Each behavior in the model is implemented by an object that represents this behavior. The mediator is used to externalize a behavior relationship between Compiler and Editor. This introduces an additional separation between a behavior and the tools which implement this behavior. One of the main advantage of this model is its maintainability and extendability. It is resistant to a behavior evolution by changing a mediators' implementation only rather than the source code of monitoring tools.

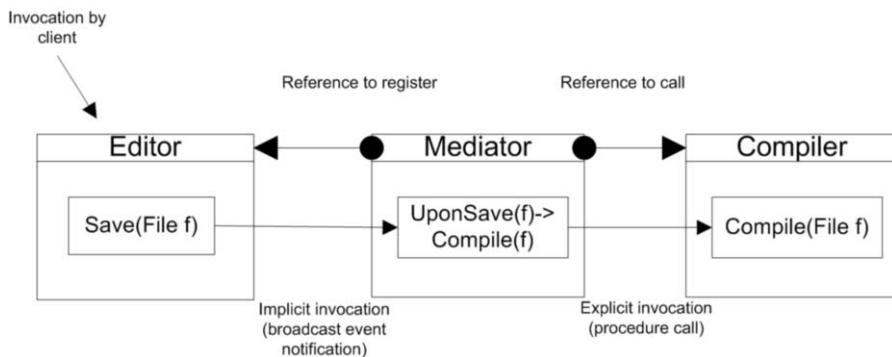


Figure 2. The mediator approach scheme The editor is viewed as an object which provides the *Save()* action and generates *Saved()* action. A notification about saving a file is done by the broadcast message. The Mediator “captures” this message and calls the compiler’s *Compile()* method.

3. OMIS vs. the Interoperability of Tools

In the context of run-time tools, by interoperability we mean the ability of applying two or more tools to the same software system, at the same time. However, in order to provide these features some common problems have to be solved, including *structural* and *logical conflicts* [2], with special attention to the *consistency* and *transparency* aspects of the logical conflicts.

Monitoring tools are usually built on top of some module that is responsible for acquiring the necessary data at run-time. This module is called *monitoring system* [9]. OMIS and the OCM monitoring system, its reference implementation, are intended to provide a kind of interoperability for tools, relating to structural conflicts and conflicts on exclusive objects [4], [5]. All tools that share an object use the same controlling monitor process that coordinates accesses to the monitored objects. Therefore, tools are enabled to coexist. However, the transparency problem (where some changes to a monitored application introduced by one tool should be invisible for another tools) was not resolved yet. The support for avoiding logical conflicts in OMIS is incomplete.

The concept underlying the original OMIS allowed to extend OMIS and the OCM, by monitoring support for Java distributed applications, known as J-OMIS and J-OCM, respectively [7]. To provide a wide interoperability for Java-oriented tools, we have designed Java Interoperability EXTension (JINEXT) as an extension to OMIS (see Figure 3).

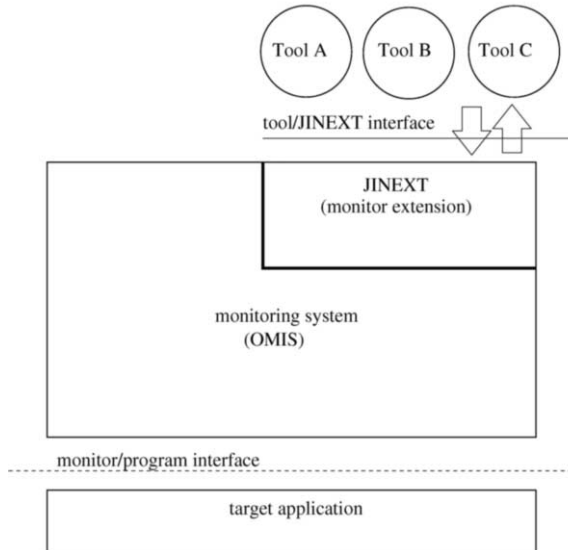


Figure 3. The overview on JINEXT architecture

Each tool intended to cooperate with JINEXT must implement a single *interface* called *tool type*. It is defined by *actions* which can be executed by the tool implementing the interface and by the *events*, which can be generated by this tool. Interfaces are used to emphasize that the behavior and functionality provided by each tool is the same for all the tools of a given type, even if provided by different vendors, written in different technologies.

The *mediators* in JINEXT follow the *mediator* interoperability model [3]. A mediator is a link between the event generated by a tool and the action which should be executed by another tool, whenever the event occurs. In JINEXT each mediator is used to describe one *behavioral relation* between an event and an action.

If used, a component implementing the JINEXT specification (a.k.a. *JINEXT compliant system*) is responsible for passing requests to the monitoring system and for synchronization between different tools. It is also responsible for choosing target monitoring tools to be informed about changes, due to request submitting. It means that tools communicate with the extension only, instead of direct calls to the monitoring system. Requests sent by tools to the extension are the same as if they were sent directly to the monitoring system.

4. Use Case

In this section we present an example of using JINEXT and its prototype implementation, JINTOP. It describes how JINTOP can be used in order to provide the interoperability of monitoring tools. An *interoperability scenario* is as follows. The user changes the source file of an example application *App*. After the **editor** *saves* the changes **compiler** *recompiles* the code and generates a new binary file. The simultaneously used **debugger** *restarts* the debugged application with the new code. During the restart of the application **profiler** *suspends* and *resumes* the previously defined performance measurements.

Thanks to mediator model used in JINTOP, we can focus on the behavioral relationships between tools, by discovering events and actions provided by each tool, as well as relations between them neglecting tools' implementations. In Figure 4 a graph of behavioral relationships for our test scenario is presented. There are four mediators used to provide interoperability between the monitoring tools.

The example monitoring tools: an editor, a compiler, a debugger and a profiler were written in C language with the GUI provided by the GTK toolkit.

4.1. Messages Exchanged During the Scenario

In order to present the benefits of using JINTOP (or any other JINEXT compliant module) we analyzed messages sent between monitoring tools and/or monitoring system for an example scenario. Both refer to the *save()* operation executed in the *editor* tool.

For the first case, where tools worked without JINTOP, we have noticed that there are a lot of messages exchanged between the monitoring tools. Some of them are unnecessary (for example a `source_code_modified()` sent by *Editor* to *Debugger*). It was sent because the tools do not know about the messages in which other tools are interested, so they have to send every message to each tool. There is a lot of `notify_about_me()` messages sent at the beginning of the sequence. These messages are necessary if we want each tool to know about other tools present within the monitoring environment. Each tool receives each message and has to decide if it is interested in this message. So, CPU load grows up. As we have observed JINTOP causes that less messages are exchanged between the monitoring tools and the monitoring system. The CPU usage and the network load (if messages are sent via

network) are reasonably lower. JINTOP decides on which messages should be sent to which tools. It makes the evolution of a system simpler and faster.

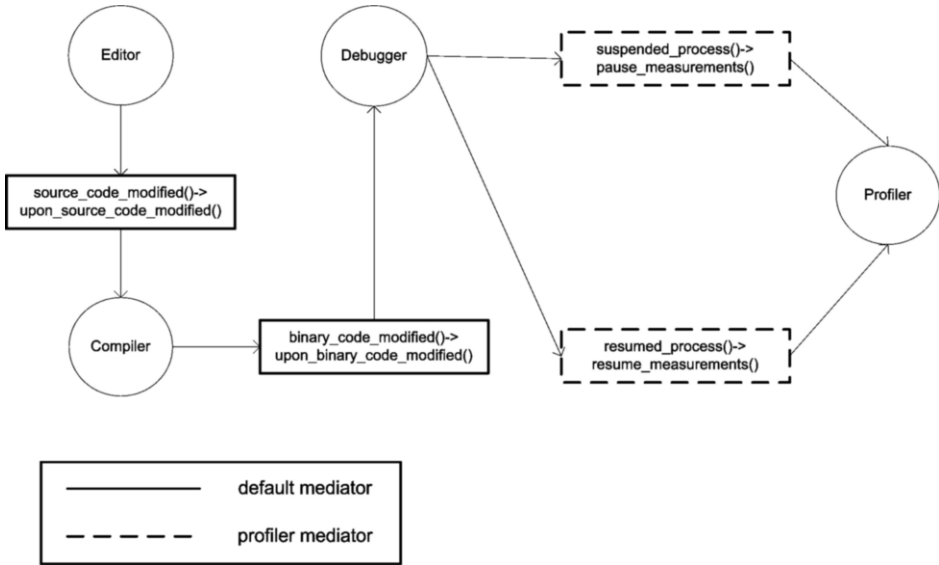


Figure 4. The behavioral relationship graph for the example tools

4.2. Influence on Profiler Measurements

In order to present the advantages of using JINTOP we have carried out the following experiment. An example profiler tool is used to measure how much time an example application *App*, which is observed by the profiler, spends in the `test` routine which is executed every 1, 2, 4, 8, 12, 8, 6, 3, 2, 1 seconds. This execution time of the `test()` routine was measured by the profiler tool. In one case the profiler worked as the only tool attached to the application. In the second and third case, the profiler operated on the application together with the editor, compiler and debugger, working on the application or its source code. In the second and third scenario, the debugger tool was used to suspend and resume the execution of the application for a random interval. Only in the third case JINTOP was used to inform the profiler tool to suspend the measurements whenever the debugger stopped the *App* application.

Figure 5 presents the execution time of the `test()` routine from an example application, in function of the tick's number. Measurements were done by the profiler tool. The timings in the 2nd chart differ from the timings presented in the 1st chart. The presence of JINTOP (3rd figure) makes the timings the same as the timings from the initial measurements. When JINTOP is not used the profiler tool tends to return random values. The difference between the proper value and the value measured by the tool is illustrated in Figure 6. As we can see the lack of a coordinating facility, i.e. JINTOP, causes that the tool returns random values. The reason is that profiler is not aware if and when the measured application is suspended.

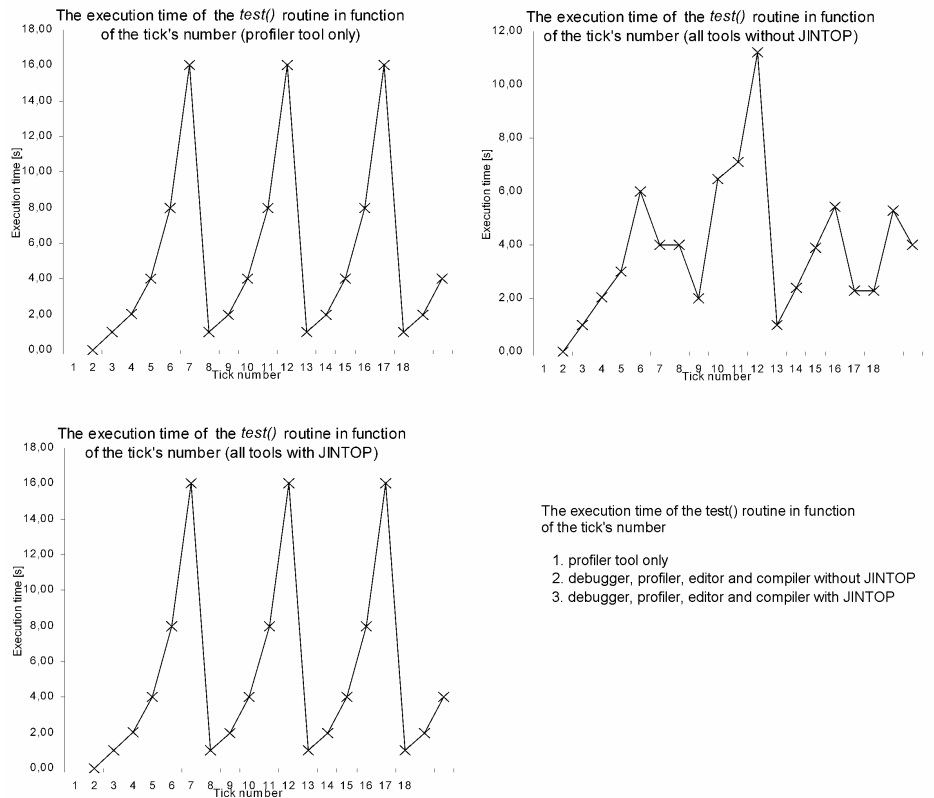


Figure 5. The use case timings for the profiler attached to the application

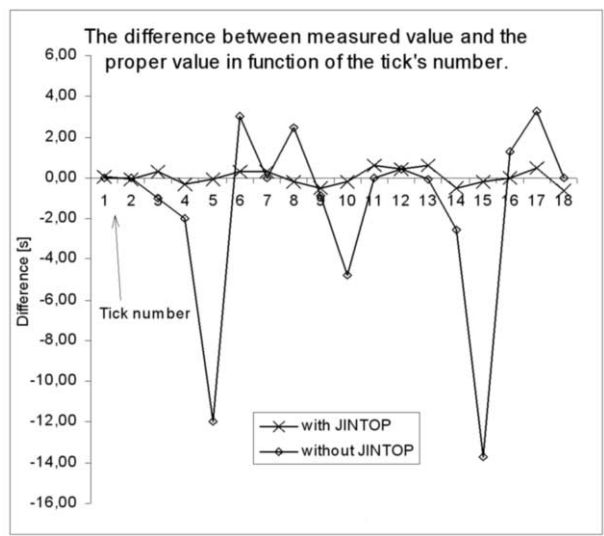


Figure 6. The difference between the measured value and the proper value in function of the tick's number

4.3. Overhead due to Use of JINTOP

To provide reliable monitoring data, the overhead introduced by the monitoring system should be as little as possible, especially when a run-time environment is observed (in contrast to the development phase, when overhead is not so critical issue). We have carried out experiments to measure the overhead induced due to the use of JINTOP instead of direct calls to the OCM. The results of our experiments are presented in Figure 7. The experiment was done on AMD Athlon 2000+ 1,8GHz with 1GB of RAM, under Linux 2.23 OS.

The execution time when using JINTOP, if there are no monitoring tools registered, is only about 3% higher than if direct OCM requests are used. This overhead results from parsing the request sent by a tool. The overhead grows up as the number of tools registered to JINTOP increases but is less than 30% and still can be reduced by changing the internal data structures of JINEXT implementation (e.g. by using hash maps instead of lists etc.).

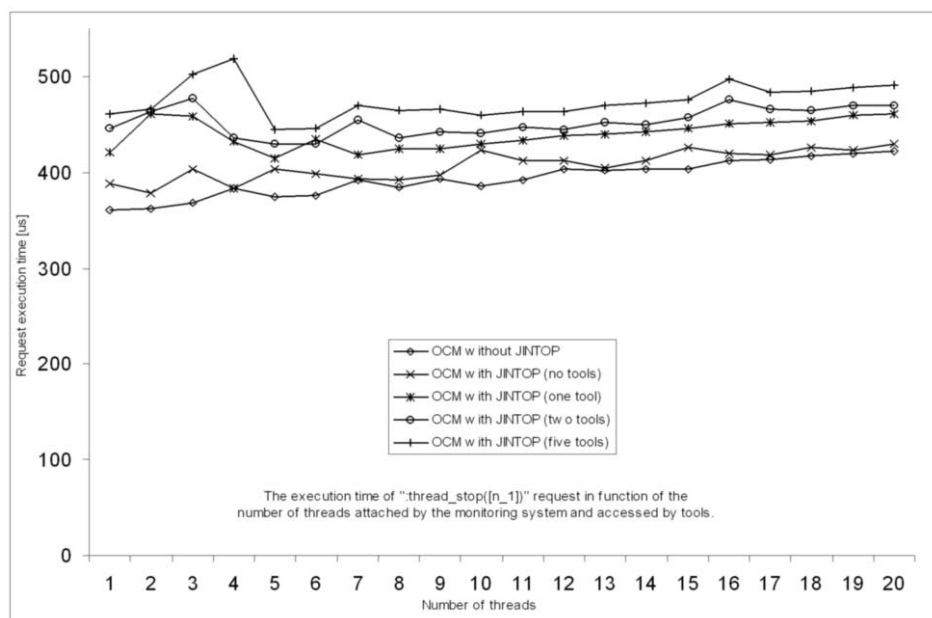


Figure 7. The execution time of an example request in function of the number of threads attached to the monitoring system vs. the number of tools attached.

5. Conclusion and Future Work

In this paper we presented JINEXT, an extension to OMIS which provides the interoperability of monitoring tools in it. The proposed solution extends the commonly known mediator model what guarantees that an evolution of the behavior of monitoring tools can be done seamlessly from the viewpoint of interoperability.

We have shown the advantages of using JINTOP with some experiments. It was proved that JINTOP guarantees the proper behavior of monitoring tools (example profiler tool), reduces the number of messages exchanged between monitoring system and/or monitoring tools. We also measured the overhead due to using JINTOP. It seems that a further evolution of the system should consider an issue of reducing the overhead when many tools are attached to the system. Some useful services should also be added to JINEXT, e.g. those ones which make it possible to dynamically add a new tool type, allowing the developer to describe the monitoring tool type in an external XML file. The next step is to extend JINTOP to be compatible with OCM-G [8], to work in grid environments. A solution would be to add a special interoperability agent on each node of the grid. The agent analyzes requests and recognizes whether they arrive from a local or remote node of grid. In the second case, the agent will do all necessary tokens translations. The agent can also support an additional interoperability-related security policy.

Acknowledgements

We are very grateful to prof. Roland Wismüller for valuable discussions. This research was partially supported by the KBN grant 4 T11C 032 23.

References

- [1] Bergstra, J. A., Klint P.: The ToolBus – a component interconnection architecture, Programming Research Group, University of Amsterdam, Meeting, Band 1697 aus Lecture Notes in Computer Science, page 51–58, Barcelona, Spain, September 1999. <ftp://info.mcs.anl.gov/pub/techreports/reports/P754.ps.Z>
- [2] Roland Wismüller: Interoperable Laufzeit-Werkzeuge für parallele und verteilte Systeme, Habilitationsschrift, Institut für Informatik, Technische Universität München, 2001
- [3] K. J. Sullivan. Mediators: Easing the Design and Evolution of Integrated Systems. PhD. thesis, Dept. of Computer Sciences and Engineering, Univ. of Washington, USA, 1994. Technical Report 94-08-01. <ftp://ftp.cs.washington.edu/tr/1994/08/UW-CSE-94-08-01.PS.Z>
- [4] Ludwig, T., Wismüller, R., Sunderam, V., and Bode, A.: OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series, (1997) <http://www.bode.in.tum.de/omis/OMIS/Version-2.0/version-2.0.ps.gz>
- [5] Roland Wismüller: Interoperability Support in the Distributed Monitoring System OCM, In: R. Wyrzykowski et al., (eds.), *Proc. 3rd International Conference on Parallel Processing and Applied Mathematics – PPAM'99*, pages 77-91, Kazimierz Dolny, Poland, September 1999, Technical University of Czestochowa, Poland. Invited Paper.
- [6] Katia Sycara, Jianguo Lu, Matthias Klusch: Interoperability among Heterogenous Software Agents on the Internet, The Robotics Institute Carnegie Mellon University, Pittsburgh, USA, October 1998
- [7] M. Bubak, W. Funika, M. Smętek, Z. Kiliański, and R. Wismüller: Architecture of Monitoring System for Distributed Java Applications. In: Dongarra, J., Laforenza, D., Orlando, S. (Eds.), *Proceedings of 10th European PVM/MPI Users' Group Meeting*, Venice, Italy, September 29 - October 2, 2003, LNCS 2840, Springer, 2003
- [8] Baliś, B., Bubak, M., Funika, W., Szipieniec, T., and Wismüller, R.: An Infrastructure for Grid Application Monitoring. In: Kranzlmüller, D. and Kacsuk, P. and Dongarra, J. and Volkert, J. (Eds.), *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 9th European PVM/MPI Users' Group Meeting, September – October 2002, Linz, Austria, 2474, Lecture Notes in Computer Science, 41-49, Springer-Verlag, 2002
- [9] Wismüller, R.: Interoperability Support in the Distributed Monitoring System OCM. In R. Wyrzykowski et al., editor, *Proc. 3rd International Conference on Parallel Processing and Applied Mathematics – PPAM'99*, pages 77-91, Kazimierz Dolny, Poland, September 1999, Technical University of Czestochowa, Poland. Invited Paper.

TCPN-Based Tool for Timing Constraints Modelling and Validation

Sławomir SAMOLEJ^a and Tomasz SZMUC^b

^a *Computer and Control Engineering Chair, Rzeszów University of Technology,
ul. W. Pola 2, 35-959 Rzeszów, Poland,
e-mail: ssamolej@prz-rzeszow.pl*

^b *Institute of Automatics, AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland,
e-mail: tsz@agh.edu.pl*

Abstract. A new software tool for hard real-time system timing constraint modelling and validation is presented. The tool consists of a set of predefined timed coloured Petri net (TCPN) structures. The structures include built-in mechanisms, which detect missing timing constraints and make it possible to validate the timing correctness of the modelled system. The final model of the system is a hierarchical TCPN following the execution and simulation rules of CPN/Design software. The paper focuses on presenting the construction of the software tool and the guidelines for applying it in real-time system development.

Introduction

One of systematically developed branches of software engineering is the application of formal languages to modelling and verification of real-time systems represented at the application level (i.e. consisted of a set of mutually cooperating concurrent tasks). Among well-established formal real-time systems specification and analysis languages (such as time/timed process algebras, real-time logics and time/timed Petri nets [3]) only high-level time/timed Petri nets [11], [7], [6], have been successfully implemented in the modelling and validation of systems at the aforementioned level [5], [12], [11], [15], [13]. The most important reasons why the application of high-level time/timed Petri nets is successful are as follows. First, the nets have the sufficient expression power to concise model complex systems without losing precision of detail. Moreover, the built-in time model they possess can represent (hard real-time) timing constraints of the software [14]. This paper presents the continuation of the abovementioned research area and concerns the application of timed coloured Petri nets (TCPNs) [7] to real-time systems development.

The results presented in this paper are an extension of the preliminary hard real-time systems timing modelling and verification method explained in [15], [13]. The main improvements to this method were in the implementation of immediate ceiling priority resource access protocol [16], [1], the introduction of external interrupts modelling, the refinement of both a scheduling module and a model integration procedure. The method was implemented in Design/CPN [10] software tool as a library of predefined TCPN structures.

The main concept of the method lies in the definition of reusable timed coloured Petri nets structures (patterns). The patterns make it possible to model timing properties and interactions of typical components in a real-time system such as tasks, resources, scheduling modules, and interrupt sources. The processor time consumption, fixed priority scheduling, task synchronization and communication can be naturally modelled. The analysis of hard real-time software timing properties is of primary interest here. Therefore, the system's functional aspects are omitted, and it is assumed that the structure of the system is described as a set of scheduled, cooperating real-time tasks (see [1], [2], [4]). The resulting model of the hard real-time system is a hierarchical TCPN composed of the patterns. Some of the predefined TCPN structures were equipped with a built-in time-out detection mechanism that made it possible to detect any missing timing constraints in the modelled system and could have been used to validate system's timing correctness. As a result, an executable timing constraints TCPN-based model of real-time application can be constructed and validated by using both simulation and state space analysis.

This paper is organised as follows. Sections 1 and 2 describe the refined patterns, which make it possible to model the core elements of real-time software, such as the task and scheduler. Section 3 presents the set of model integration rules and includes a case study. Conclusions and a future research programme complete this paper.

It has been assumed that the reader is familiar with the basic principles of TCPN theory [7]. A thorough study of hard real-time system timing properties can be found in [2]. Practical aspects of real-time systems programming is included in [1].

1. Real-Time Task Modelling

The central component of the real-time timing constraints modelling and verification tool, which is proposed, is the task model. Any task may interact with the other tasks. Moreover, any task can be executed by following certain fixed priority scheduling policy. The task structure is divided into two layers. The first layer (the main task model, see Section 1.1) enables modelling of interfaces between the task and other system components, i.e. scheduler, resources, and task initiation modules. Additionally, the main task model includes a built-in automatic detecting policy that allows the monitoring whether the task meets its timing constraints. The second layer includes a detailed task execution model with shared resources accessing protocol taken into account.

The state of each task in the system is represented by a tuple TCB (*Task Control Block*): $TCB = (ID, PRT, STATE, UTIL, TCNT, RES_NAME, S_PRT, D_PRT)$, where *ID* is *task identifier*, *PRT* is *current task priority*, *STATE* is *state of a task*, *UTIL* is *task utilisation factor*, *TCNT* is *cycle counter*, *RES_NAME* is *resource name* on which the task is currently blocked (waiting for allocation), *S_PRT* is *task basic static priority*, and *D_PRT* is *task dynamic priority*. Task identifier is a unique task name (e.g. Task1, Task2,...). *Current task priority* is an integer value that is taken into account during the execution of scheduling policy. The *state of the task* is of symbolic type, where its individual values correspond to typical task states (in real-time operating system terminology [2]): *Running*, *Ready*, *Idle*, *Blocked*, *NotInit*, *Timeout_Err*. *Task utilisation factor* indicates the actual state of the task execution progress. The initial value of the factor is set on the worst-case execution time (WCET) of the modelled task. The successive reduction of the factor takes place during the execution of task computations.

When the factor reaches a value of zero that means that the current instance of the task has finished its execution. *Cycle counter* may be used for statistical analysis and has no influence on the net states. *Task basic static priority* is the priority, which is assigned according to the fixed priority scheduling strategy. *Task dynamic priority* includes the task priority value, which becomes the current task priority during the resource access protocol execution.

The aforementioned tuple was implemented in CPN ML [10] language as colour set [7] in the following way:

```
color ID = with Task1 | Task2 | Task3 | Task4 | Task5 |
              Task6 | Task7 | Task8;
color PRT = int;
color STATE = with Ready | Running | Idle | NotInit |
                Timeout_Err | Crit_Sect | Blocked;
color UTIL = int;                color TCNT = int;
color D_PRT = int;                color S_PRT = int;
color RES_NAME = with No_Res | Res1 | Res2 | Res3 | Res4;
color TCB = product
ID*PRT*STATE*UTIL*TCNT*RES_NAME*S_PRT*D_PRT timed;
```

The general state of a modelled system can easily be computed by monitoring the task's *TCBs*.

1.1. Main Task Model

The main TCPN-based model of an example task is shown in Figure 1. The current state of the task is represented by *Task_st* place marking. The place includes one token *TCB* type, whose state can only be changed by firing *Act_task* or *Run_task* transitions.

Act_task transition fires after a new token in *Init_Port* place appears. Firing *Act_task* transition initialise of a new task instance in the system. Thereafter, the new task instance acknowledges that it should be taken into consideration in the scheduling policy (by placing a token to *To_Ready* fusion place) and verifies whether the previous task instance has already completed its computations. The *task_timeout_check()* function attached to one of output arcs of *Act_task* transition interprets the current state of the task as follows:

```
fun task_timeout_check(task_TCB:TCB, sys_st: SYS_ST)=
if
  (#3 task_TCB = Ready      orelse
   #3 task_TCB = Running    orelse
   #3 task_TCB = Blocked    orelse
   #3 task_TCB = Crit_Sect)
then
  case #1 task_TCB of
    Task1 => 1`Tsk1_err | Task2 => 1`Tsk2_err |
    Task3 => 1`Tsk3_err | Task4 => 1`Tsk4_err |
    Task5 => 1`Tsk5_err
  else
    1`sys_st;
```


If the previous instance of the task has already finished its computations, the new instance reaches a *Ready* state and waits for processor and resource access. If the previous instance has not finished, an error occurs. Detection of an error at this stage implies the suitable modification of the token in *Sys_st* fusion place. Consequently, the entire system model reaches a dead marking. Additionally, during the task initialisation process the current state of token in *In* port place is transferred to *Proc_Sig* place. The migration of *SIGNAL*-type tokens (belonging to places *In*, *Proc_Sig* and *Out*) constitutes the additional external time-out detection mechanism. This mechanism makes it possible to control timing parameters of interrupt services in the system model and is presented in detail in next sections.

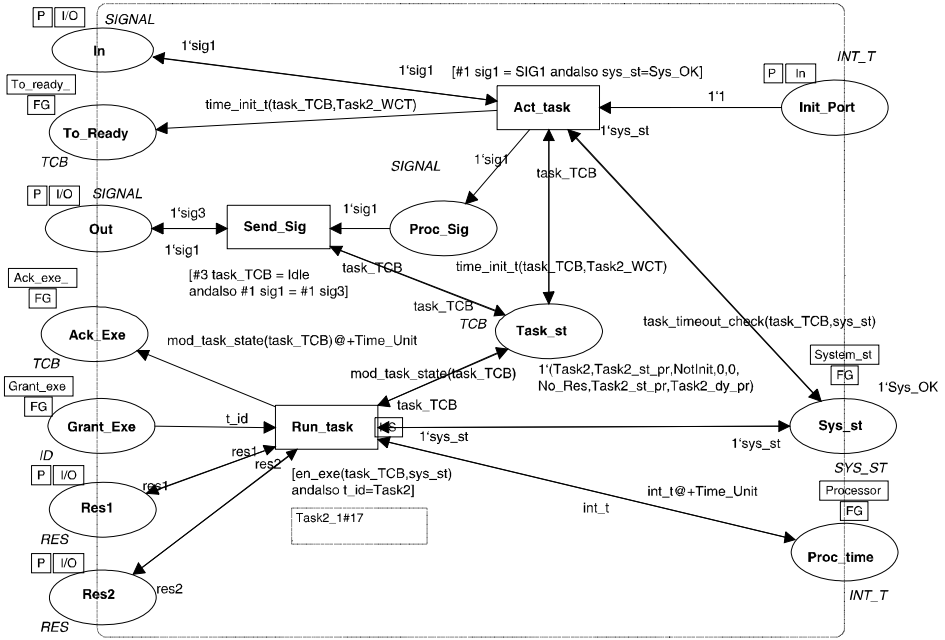


Figure 1. Main task model

After completing the task initialisation, the execution of the current task instance is possible. As the set of tasks attached to one processor runs according to fixed priority scheduling policy, any individual task may continue its computation only if it obtains permission from the scheduler. Fusion places *To_ready*, *Grant_Exec* and *Ack_Exec* constitute the interface between the scheduler and the set of tasks. Additionally, the set of tasks executed on one processor shares one *Proc_time* fusion place. Any task execution may also be restricted by resource accessibility. The resources that are used during the execution of the task are represented by tokens attached to the resource (*RES*) places (compare *Res1* and *Res2* places in Figure 1). *Run_task* substitution transition in Figure 1 connects the main task model to a separate subnet, which includes the execution task model discussed in the next section.

This briefly outlined pattern task structure ought to be appropriately adjusted to obtain a functional individual task model. First, basic task properties should be

established by setting the initial marking of *Task_St* place (the consecutive components of task *TCB* tuple should be defined: individual task identifier, initial priority, worst-case execution time, relative deadline, resources used, basic priority, and dynamic priority). Finally, the *Act_Task* transition guard function ought to be adequately adjusted (#1 *sig1* parameter should be equal to adequate *SIGn* name of signal processed). To sum up, the main task model constitutes a generic framework, which makes it possible to set up the individual task properties and port the task to the other components of the system (scheduling module, processor, task initiation subsystem).

1.2. Execution Task Model

The execution task model makes it possible to precisely define the subsequent task computation stages. Both the scheduling and the resource access policy (which are implemented in the TCPN-based software tool) can influence on the stages. The execution task model pattern was implemented as a separate subnet, which can be attached to *Run_task* (Figure 1) substitution transition. The structure of the subnet may be of the shape shown in Figure 2. Execution of the task has been divided into separate phases represented by individual transitions. The execution of one quantum of computations is modelled as a firing of one of the aforementioned transitions with external scheduling policy and resource accessibility taken into account. The scheduling policy assigns the processor time to a task by placing a token in *Grant_exe* place. Subsequently, the task processes a quantum of computations by firing the corresponding *Run_T_Sn* transition, and finally places a token in *Ack_exe* place. The appearing a token in *Ack_exe* place means that the task completed the quantum of computations. Selection of the transition depends on the actual state of task utilisation factor (*UTIL* field in *TCB* tuple) monitored in each transition guard function. The execution of one quantum of computations causes the corresponding modification of a time stamp to be attached to the token in the *Proc_Time* place. Simultaneously, a corresponding reduction of *UTIL* element of *TCB* task tuple (see Section 1) occurs. Consequently, the processor time consumption can be modelled.

Some computation phases cannot proceed without access to an appropriate resource. For example, the task in Figure 2 requires access to *Res1* resource during the first stage of its computations and to *Res2* resource during the second. The allocation (*take_res()* arc function) and release (*release_res()* arc function) actions (events) are modelled as shown in the net. The resource access proceeds according to the immediate ceiling policy [16], [1] as follows:

- Each task has a static default priority assigned (*S_PRT* field in *TCB* tuple).
- Each resource has a static ceiling value defined, this is the maximum priority of the processes that use it.
- A task has a dynamic priority (*D_PRT* field in *TCB* tuple) that is the maximum of its own static priority and ceiling values of any resources it has blocked.
- The task reaches its dynamic priority (*D_PRT*) at the resource access and returns to its static default priority (*S_PRT*) after resource release.
- The scheduler (see next section) manages several tasks having the same priority level by taking their arrival time in account. The later the task arrives, the later it proceeds.

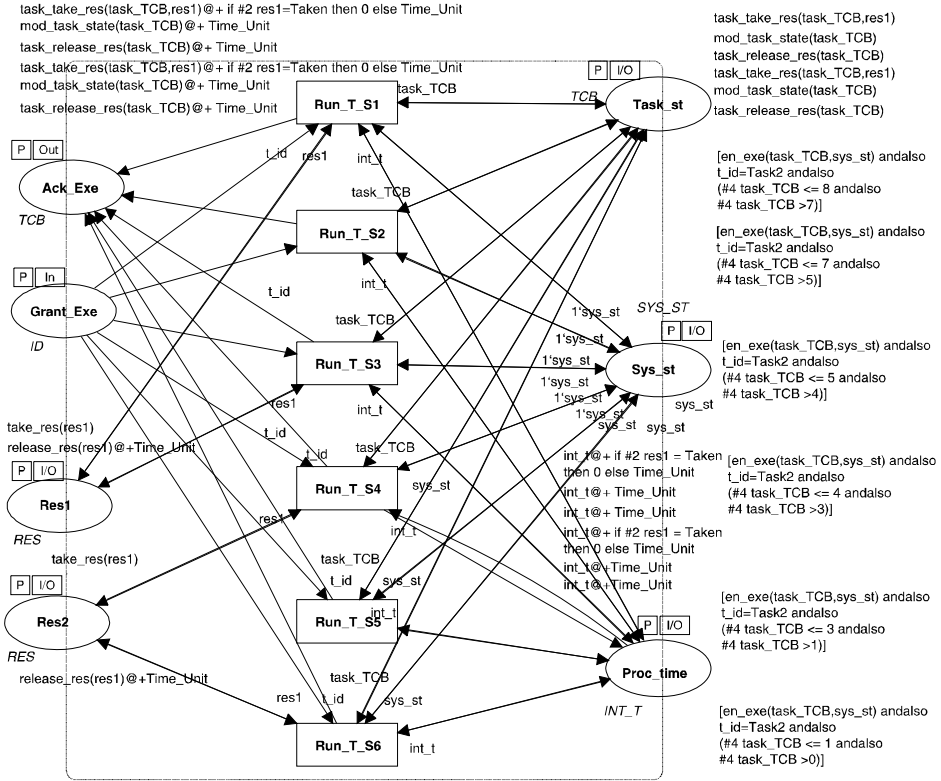


Figure 2. Execution task model

The following example arc expression CPN ML code presents the modification of a *TCB* during the first stage of resource access:

```
fun task_take_res(task_TCB: TCB, res:RES)=
  if #4 task_TCB > 1 andalso #2 res = Free
  then ( #1 task_TCB,
          #8 task_TCB, (*enter dynamic ceiling priority*)
          Running,
          #4 task_TCB - Time_Unit, (*modify UTIL*)
          #5 task_TCB, No_Res, #7 task_TCB,
          #8 task_TCB)
  else
    if #4 task_TCB > 1 andalso #2 res = Taken
    then ( #1 task_TCB,
            #2 task_TCB, Blocked, #4 task_TCB,
            #5 task_TCB, #1 res, #7 task_TCB,
            #8 task_TCB)
    else
      ( #1 task_TCB, #2 task_TCB, Idle,
        #4 task_TCB - Time_Unit, #5 task_TCB,
        No_Res, #7 task_TCB, #8 task_TCB );
```

The execution task model involves a modelling of the task computation (processor time consumption), resource management, process synchronization, and synchronous and asynchronous inter-task communication.

1.3. Task Initialisation

The initialisation of the task computations, which was briefly introduced in Section 1.1, starts after an appearance of the corresponding token in *Init_port* place (see Figure 1). Both an event-driven (e.g. an external interrupt) and a time driven initialisation of the task computations can be applied in the software tool developed. A separate initialisation module (subnet) can be attached to each task model. Figure 3 includes a pattern of the module.

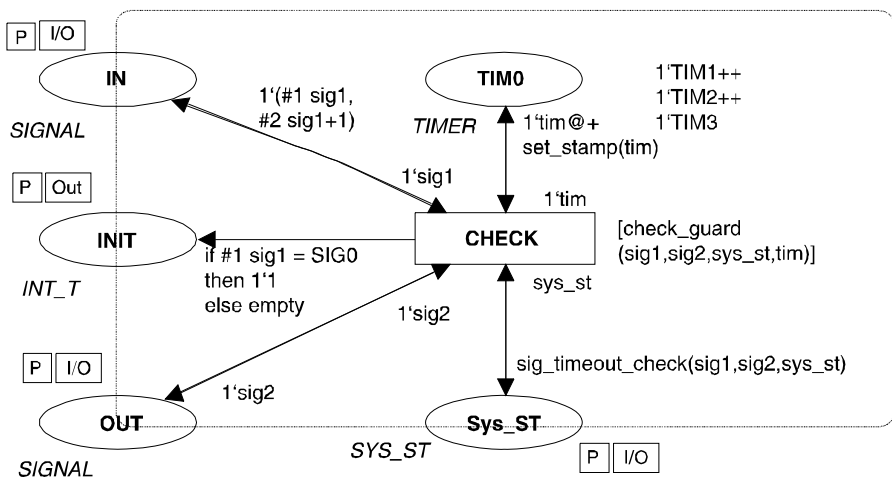


Figure 3. Initialisation module

The core of the module consist of *TIMO* place and *CHECK* transition, both constituting a timer that can periodically or sporadically (by using Design/CPN implemented random number functions [8]) modify the state of tokens in *IN*, *OUT*, *INIT* and *Sys_ST* places. The *INIT* port place should be connected to corresponding *Init_port* place in the main task model (compare Figure 1), and the appearance of the token in it causes the periodic or sporadic task initialisation process. Similarly, *IN* and *OUT* port places ought to be attached to the corresponding *In*, and *Out* places in the main task model. The states of the tokens in the mentioned places are modified at the beginning and at the end of task computations. The monitoring of the states makes it possible to detect whether the task has finished the computations before the stated deadline. Missing the deadline causes the appropriate modification of *Sys_st* place marking and reaching a dead marking of the whole system model. Hence, the initialisation module constitutes another timing correctness detection mechanism in the TCPN-based software tool. The mechanism is particularly usable for the timing validation of sporadic and periodic task with deadlines less than period. During the modelling of the periodic task with deadlines equal to the period, the state of the tokens

in the queue represented by *ExeQueue* place. The queue located in the *ExeQueue* place includes currently activated tasks, which are ordered with respect to task priorities. Tasks having the same priorities (due to the immediate priority ceiling procedure implemented) are ordered according their arrival times. If an instance of task has completed its computations, it is removed from the task queue.

To sum up, the scheduler activity consists of the following actions: the scheduler accepts new activated task instances, schedules the set of tasks according to a settled fixed priority algorithm, allocates processor time to the selected task instance, and monitors the actual states of the task instances. The task priorities may dynamically change due to the priority ceiling protocol implemented. The task selection proceeds cyclically according to the settled processor cycle time. In every cycle, pre-emption of the currently executed task is possible.

3. Model Integration (Case Study)

A set of TCPN structures (patterns) presented in previous sections makes it possible to construct a timing model for a typical hard real-time application. The model can be tested with the meeting timing constraints restriction taken into account. An example application of the patterns for modelling a control system is shown in this section. The specification of the system is given in Table 1.

Table 1. Example control system specification

Proc. Cycle	Task	Task Per.	Task WCET	Task Dyn. Prt.	Task Stat. Prt.	Resources									
1	Task1	(10-20)	4	3	3	1		2	3	4					
								Res1							
	Task2	50	8	2	3	1	2	3	4	5	6	7	8		
						Res1				Res2					
	Task3	90	10	1	2	1	2	3	4	5	6	7	8	9	10
						Res2									

The system is run on one processor and every quantum of computation takes one unit of time. Three hard real-time tasks with assigned attributes (i.e.: dynamic priority, static basic priority, period or estimated bounds of period for a sporadic task (for *Task1*), and *WCET*) constitute the core of the model. Moreover, tasks: *Task1* and *Task2* share resource *Res1*, and *Task2* and *Task3* share resource *Res2*.

The corresponding timed coloured Petri net models have been prepared on the basis of the above specification. The nets in Figures 1 and 2 follow the specification of *Task2*. The *Task2* initiation module and the scheduler model for the system is presented in Figure 3 and Figure 4, respectively. Models of the remaining system components can be constructed accordingly (by adequate modifications of the TCPN-based patterns).

After preparing the complete set of subnets expressing the corresponding timing features of all specified components of the system, these subnets can be connected by means of one overall net as shown in Figure 5.

The final model of the system is a hierarchical TCPN, which follows the execution and simulation rules of CPN/Design software tool. The basic timing validation technique available in the software tool discussed in the paper is simulation. The built-in missing timing constraints detection mechanism causes that the model under simulation reaches a dead marking if the missing occurs. Therefore, the schedulability of the set of task (with resource sharing taken into account) can be automatically validated. When the simulation tool detects a dead marking, the net state is automatically stored. This makes it possible to deduce the cause of the missing of timing constraints. Moreover, thanks to Design/CPN performance module [8] the subsequent states of the simulated system can be stored in the appropriate files and analysed offline.

Additionally, it was proved in [15] that for a certain subset of hard real-time systems the model could be fully verified by means of state space analysis of a partial reachability graph. The formal timing verification may be possible if all tasks in the modelled system have bounded and measurable blocking times, they are scheduled according RM algorithm, and the generation of the graph starts in the critical instant for the task with the lowest priority (according to theorem 1 formulated in [9]). Considering the above-mentioned assumptions it is possible to generate a partial reachability graph of the model that includes all system states during the WCET of the task with the lowest priority. If the reachability graph does not include any dead markings within the WCET, the set of real-time tasks is schedulable, and consequently the model meets all the timing constraints. Therefore, for a reasonably bounded amount of tasks (due to software tool limitation) it is possible to prove the schedulability, even if processor utilisation factor exceeds the last upper bound specified in ([9] – theorem 3).

4. Conclusions and Future Research

The paper presents the main features of the proposed TCPN-based software tool making it possible to construct and validate timing attributes of typical hard real-time applications. The main concept of the tool lies in the definition of reusable TCPN structures (patterns) involving typical components of real-time systems at the application level of development (i.e. tasks, scheduling and task initialisation modules). Additionally, some of the patterns have been equipped with missing timing constraints detection mechanisms that transfers the net state into a dead marking if the missing occurs. The software tool makes it possible to construct any model of the hard real-time software consisting of a set of mutually cooperating tasks attached to a set of processors. The final model of the system is a hierarchical timed coloured Petri net. Both simulation and state space analysis techniques can be applied to validate the timing correctness of the modelled system.

The TCPN-based software tool has been developed on the basis of the research results concluded in [15] and partly presented in [13]. However, the toll development has opened new research threads, which will centre on the following subjects. First, the dynamic priority scheduling algorithms models (e.g. EDF) with corresponding resource access protocols (e.g. SRP) will supplement the current set of available design patterns. Second, the state space analysis methods of a dynamic priority scheduled system will be investigated.

References

- [1] Burns, A., Wellings, A.: Real-Time Systems and Programming Languages, *Pearson Education Limited* (2001)
- [2] Buttazzo, G. C.: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, *Kluwer Academic Publishers* (1997)
- [3] Cheng, A.M.K.: Real-Time Systems, Scheduling, Analysis, and Verification, *John Wiley & Sons* (2002)
- [4] Douglass, B. P.: Doing Hard Time, Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, *Addison-Wesley* (1999)
- [5] Felder, M., Pezze, M.: A Formal Design Notation for Real-Time Systems, *ACM Trans. on Soft. Eng. and Met.*, Vol. 11, No. 2, (2002), 149-190
- [6] Ghezzi, C., Mandrioli, D., Morasca, S., Pezze, M.: A Unified High-Level Petri Net Formalism for Time-Critical Systems, *IEEE Trans. on Soft. Eng.*, Vol. 17, No. 2, (1991), 160-172
- [7] Jensen, K.: Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use, Vol. 1-3, *Springer*, (1996)
- [8] Linstrom B., Wells L.: Design/CPN Performance Tool Manual, CPN Group, University of Aarhus, Denmark, (1999)
- [9] Liu, C.L., Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *Jour. of the Assoc. for Comp. Machinery*, Vol. 20, No. 1(1973), 46-61
- [10] Meta Software Corporation: Design/CPN Reference Manual for X-Windows, *Meta Software* (1993)
- [11] Naedele, M.: An Approach to Modeling and Evaluation of Functional and Timing Specifications of Real-Time Systems, *The Journal of Sys. and Soft.* 57, (2001), 155-174
- [12] Nigro, L., Pupo, F.: Schedulability Analysis of Real Time Actor Systems Using Coloured Petri Nets, Concurrent Object-Oriented Programming and Petri Nets, *LNCS 2001*, (2001), 493-513.
- [13] Samolej, S., Szmuc, T.: Time Constraints Modelling and Verification Using Timed Coloured Petri Nets, Real-Time Programming – WRTIP'04 IFAC/IFIP Workshop, Istanbul, Turkey, 6 – 9 September, (2004), (to appear)
- [14] Samolej, S., Szmuc, T.: Time Extensions of Petri Nets for Modelling and Verification of Hard Real-Time Systems, Computer Science, *Annual of University of Mining and Metallurgy*, Kraków, Vol. 4, (2002), 55-76
- [15] Samolej, S.: Design of Embedded Systems using Timed Coloured Petri, Ph.D. Dissertation (in Polish), AGH University of Science and Technology, Cracow, Poland, (2003)
- [16] Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: An approach to real-time synchronisation, *IEEE Transactions on Computers*, Vol. 39 (9), (1990), 1175-1185

This page intentionally left blank

5. Requirements Engineering

This page intentionally left blank

Requirement Management in Practice

Michał GODOWSKI and Dariusz CZYRNEK

ComArch S.A., AGH University of Science and Technology

Abstract. This paper is about software requirements, main problem that may come from incorrectly defined requirements, bad practices and common mistakes made while creating requirements. Briefly describes methodologies of managing software requirements. In the article Requirement Management System Theater (RMST) is presented, system used to work with software requirements, keeps its history and manage them. It concentrates on managing versioned software products. Rational RequisitePro and CaliberRM systems are described for comparison.

Introduction

Contracts for business transactions and software requirements specifications (SRSs) share many properties, including the need to be precise and accurate, to be self-consistent, and to anticipate all possible contingencies. SRS are usually depicted in a natural language, but in many circumstances it is not complete and requires additional diagrams, dedicated notations, formulas and special abstract languages. In most cases, the origin of all requirements is a text notation like any other request for proposal.

The research conducted at Università di Trento¹ in Italy, shows that a major part of SRS documents are composed by end-users or during interviews with future customers. Percentage participation of most common way of gathering requirements is as follows:

- 71.8% of these documents are written in common natural languages,
- 15.9% of these documents are written in structured natural language, and
- only 5.3% of these documents are written in formalised language.

The natural specification manner of any business contract and other legal documents are natural languages. Rarely information is provided in other notations, to clarify and amplify natural language text.

1. Requirement Management

Inadequate, incomplete, erroneous, and ambiguous system and SRS are a major and ongoing source of problems in systems development. These problems manifest themselves in missed schedules, budget overruns, and systems that are, to varying degrees, unresponsive to the true needs of the customer.

¹ Quoted by Università di Trento, online.cs.unitn.it

The Standish Group research shows a staggering 31% of projects will be cancelled before they ever get completed. Further results indicate that 52.7% of projects will cost 189% of their original estimates ... [5].

These manifestations are often attributed to the poorly defined and incomprehensible processes used to elicit, specify, analyse, manage, and validate requirements². There are many approaches to requirement management. Let us mention only two of them: The Capability Maturity Model for Software [1], Prince2 and FURPS+ [2].

1.1. Requirement Analysis & System Goals

The requirement engineer has to answer the following questions:

- What do *stakeholders* want from the system?
- What should it do? How should it look like? Sounds like? Be like?

The high level goals apply to overall system including:

- What will the system do and be like?
- Typically span use cases of complex system
- Each stakeholder requires some value from system (what value?)

Requirement analysis starts with a project³ description usually specified by major stakeholder:

- a.k.a. a *customer* can be an external customer, or internal customer (i.e. another department of the same company, management, professor,...)
- or project team writes it for an anticipated market

Requirement analysis usually means detailing the requirements, but still in a language that the user understands. The main goal is to complete a vision and use cases.

Requirement analysis is much more a study, rather than a project description, so that it requires:

- Interviewing users, clients, managers, sponsors, experts,...
- Learning about current practices, existing systems, business rules (general and specific)
- But it does not require becoming a domain expert

Stakeholder must approve a realm and meaning of requirements before proceeding. Such as the set of approved requirements forms a basis for contracts (if real customer), bounds a client expectations, establishes a scope of work. Additionally it also specifies what the system will *not* do, so that it narrows the focus of a project team and limits the range of system goals. It also prevents a so-called '*creep scope*' effect, when one functionality is differently understood by each side of business.

² Requirement & sub-requirement – defined as: “a condition or capability to which a system must conform”

³ Project – a set of functionality and constraints registered to a product

The purpose of collecting requirements is to explore project *feasibility*, target length (in weeks), identify most use cases and actors, make rough estimate of costs. The project description is used to comment on the system which is usually a *black box*, probably vague, repetitive, confused. The major risk and reason of misunderstanding is that – from the customer’s point of view – everything is clear and is specified.

Bad practice is to go too deep in implementation details, too limiting at this stage. It is very often that some of the requirements are rather a *wish list* without clearly goals.

1.2. Requirement Categorisation / Requirement Types

There are many categories of requirements. The main distinction is between functional and non-functional requirements. According to an essence of requirements we can also mention the following types: usability, supportability, hardware, platform dependent requirements and so on. The methodology described in FURPS+ [2] is based on such a distinction.

The reality shows that used in FURPS+ separation is often unclear and one requirement belongs to more than one category. So it is recommended to use only a narrow set of types.

1.3. Sources of Requirements

Requirements are written by one or more professionals in the software or law field, for a customer who is not in the field. Occasionally, a professional in the relevant application domain participates in the writing. Both kinds of writing require the professional to elicit information from the customer’s representatives and to observe, interview, and negotiate with them. Both require the client’s representatives to validate that the professionals have captured the customer’s intent.

The practice is far from this model, usually requirements are declared by: customers, end-users, marketplace, trends and company competition. Then they are re-organised and corrected by analysts. However, the main goal is to meet customers’ expectations and nothing more, unless there are other reasons e.g. additional profit or usability welcome in other installations.

1.4. Roadmap

Software development should have a more or less clear vision. A direction in which the product will be expanded is called *the roadmap*. The role of Product Manager is to prepare, manage and update the system roadmap. It is delimited by requirements, market trends, end-users expectations and other strategic plans. Roadmap is composed of *baselines*⁴.

1.5. Changes

A common and pervasive problem with software specifications is that of change. The business situation is permanently changing. It involves more or less important change requests to keep the system useful. Keeping up-to-date and consistent requirements’ specification and other documents is a major and ongoing problem and is the subject of

⁴ Baseline – an entity containing a subset of requirements belongs to one project

a whole branch of Software Engineering (SE). For software, change seems to be relentless all through the life-cycle.

1.6. Fixing Errors

In SE, it is known that the cost of fixing a detected error goes up exponentially with the life-cycle.

That is, an error that costs X (where X is any measure of work or resources) to fix during requirements specification, costs about $10X$ to fix during coding and about $200X$ to fix after deployment. Therefore, it pays handsomely to find errors during requirements specification, so handsomely that it is worth spending a significant fraction of the total budget on requirements specification.

Although it is not an appropriate situation, errors in specification might be identified just in implementation stage. It is a serious problem engaging project manager, product manager and usually customers. Errors can be solved by common agreement of all business parts. It should be noted down in a contract and be reflected in SRS. Requirement modification demands updating a baseline content and starting a new cycle of analysis, project stage, implementation, tests and so on. The total cost in such a case increases considerably.

1.7. Bad Practices

Domain experts indicate common mistakes which are:

1. setting the highest priority to almost all requirements
For a project manager the information, that all requirements have the highest priority is useless and such practice should be avoided.
2. accepting contradictory collection of requirements
The contradictory requirements should be identified and marked by contradiction marker⁵ during elicitation and analysis; the role of product manager is to trace and eliminate them as soon as possible
3. accepting too general requirements SCRs containing too overall specification are a probable source of problems, because customers have a wide ability to persuade its' interpretation.
4. lack of requirements
No written requirements does not imply their absence, analogous situation with incomplete collection of requirements.
5. signing a commitment before analysis and project planning is done
6. lack of traces⁶ between implementation and production requirements
Project manager of implementation should have information when the applied requirements will be realised.

The more detailed and well grained requirements are, the more easily they are controlled though the project life-cycle.

⁵ Contradiction marker – a linkage between two or more contradictory requirements

⁶ Trace – a bound between two requirements, usually between customer requirements and requirements forwarded to production

1.8. Rational RequisitePro

One of the most known CASE tool is Rational RequisitePro [10] created by IBM.

Basically there is a tree like structure for creating requirements and splitting them into separate groups and types. Each requirement type vary by attributes it have (possible choices are: List (single/multiple value), Text, Integer, Real, Date, Time, ClearQuest Integration, Url Link. Each requirement can be seen in group view, where each parameter is represented by separate column. Users can link requirements and see the connections in traceability matrix. RequisitePro let us join requirements into baselines, which, when frozen, can't be modified.

Import and export from MS Word and CVS are supported. More over history of the requirements is remembered but searching through the previous version is not so easy.

Permissions are granted to single users or to groups of users. They allow users to: read, create, update or delete resources like requirements, requirement types and connections between requirements.

RequisitePro provides ability to cooperation of multiple users to work on single project from different machines.

The product is also integrated with other Rational CASE software, like Rational Rose, Rational XDE, Rational ClearCase, Rational ClearQuest, Rational SoDA, Rational Unified Process, Rational TestManager.

Rational RequisitePro is useful tool to support requirement management. For everyone, who did work with other software made by Rational, this would be quick and easy to get used to RequisitePro.

1.9. Borland CaliberRM

Other software for managing requirements is CaliberRM made by Borland [11]. It is one of tools designed by the firm to support software engineering.

It's main features are:

- centralised repository, combined with client-server architecture of the system let users from all over the company access requirements they have permissions to read and modify
- cooperation with other tools like: Borland StarTeam, Mercury Interactive TestDirector, Microsoft Project
- dictionaries – where users can define project specific terms and where others may look for longer description of it.

As in RequisitePro requirements are collected in tree structured groups. Each requirement type have the same attributes (possible attributes are Boolean, Date, Duration, Float, Long integer, Multiple selection list, Multiple selection group list, Multiple selection user list, Multiple line text field, Single selection list, Single selection group list, Single selection user list, Single text line). Requirements may be viewed using requirement grid or traceability matrix.

Permissions are defined by profiles. Each profile may be given access to some action in the system (creating requirement, attaching the requirement to the baseline,...).

It is possible to import requirements from MS Word document and export them to MS Words, MS Access, RTF or text.

CaliberRM has advanced mechanism of generating reports. User can define the outlook of this reports.

Client of this system is MS Word dependent and does not support other platform than MS Windows.

CaliberRM system is another variant of the systems for managing requirements. It allows easy integration with other firm software. Integrated database (Starbase) build-in server, cannot be replaced might lead to difficulties.

2. The RMSTheater – Requirement Management System

As an example of practical approach, we would like to introduce the RMSTheater. This software is a proven solution, implemented in ComArch S.A. It has been in-service for about two years and currently stores several thousands of requirements while still receiving new ones.

The Requirement Management System is software which supports:

- management of versioned products
- planning functionality to next release and date of release completion
- team work, access control and discussions with respect to user role in organisation (Product manager, Project manager, Analyst, ...)
- assigning requirements to a baseline
- flexible set of requirement parameters⁷ easily adapting to specification of each project and methodology
- hierarchical view on a set of requirements

The RMSTheater system is a repository for all requirements registered to developed or planned software product. System is available to end-user as a web-based Java Applet or a standalone

The end-users of the RMSTheater system are: product managers, solution managers, customers, project managers, analysts, R&D, testers.

2.1. Requirement Management Methodology⁸

The requirement management process in the RMSTheater starts far before a project kick-off. The product manager creates a project in the RMSTheater and, if necessary, a new entry in Customers subtree. Invited customers and solution managers can register their visions and requirements in a location associated with customer requirements. Product manager consults them with R&D, client attendances and sets a proper status on each of registered requirement. Status is flexibly profiled and – according to specific situation – can be adapted. The recommended values are: *New, Unclear, Rejected, Suspended, Accepted, Forwarded to production, Implemented*.

⁷ Requirement parameter – an instance of parameter definition within a requirement

⁸ Methodology for managing requirements for versioned products

Customers requirement can also possess a priority parameter. The accepted values are: *Critical, Major, Minor, Optional*. We discourage setting all requirements to priority *Critical* value, no matter if customers or products requirements.

Then, product manager defines a product requirements which are derived from customers' requirements and place them in a product requirement's subtree. The product requirements are usually general (based on many requirements of different customers) and aggregate different points of view. During this phase, product manager should link product requirements with corresponding customer requirements (it is called Traces). Traces brings useful information, specially for implementations. According to notes presented in traces, implementation project manager is able to qualify a version of product in which a requested functionality will be implemented.

The process of requirements elicitation requires many iterations of re-reading and re-editing activities. Product manager can set flag Suspected to requirement. When the requirements are agreed, the next step is to assign them to a baseline. Due to product manager decision, a subset of requirements is binded to a baseline which corresponds to product's version/release. Project manager should accept a scope of work or perform negotiations. If necessary, product manager should re-baseline project. The fact of re-baselining is reported in a history of affected requirements.

The RMSTheater system points at these of requirements, which have not yet been assigned to any baseline. Also these which were modified and their newest version have not yet been baselined. It helps product manager to identify a new functionality and propose them to a given version/release.

In the RMSTheater projects, groups and requirements are presented in a hierarchic tree layout (see Figure 1).

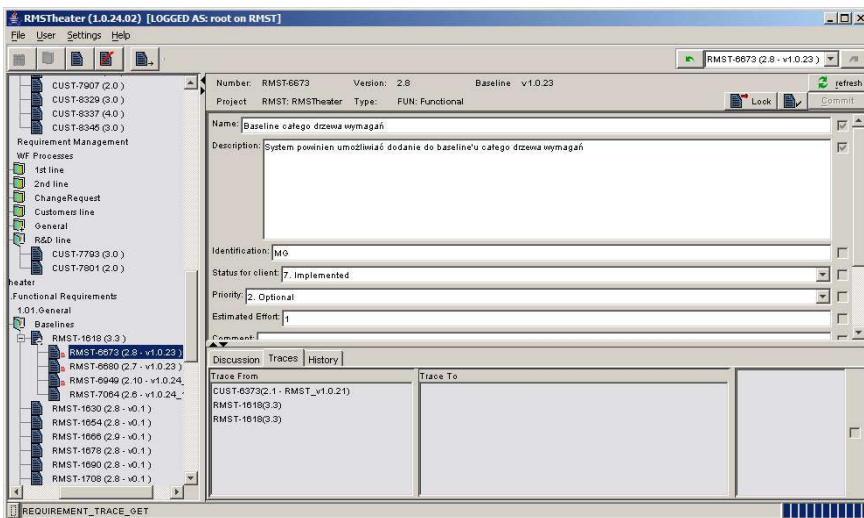


Figure 1. Projects, groups and requirements

Projects

Project is on a top of requirement structure. It holds all notes binded to a developed product. Below Projects, there are Requirement Groups or Requirements.

*Requirement Group*⁹

Each Requirement Group has a *Name*, which should be unique on each level of a tree. Additionally, it can possess a *Shortname*, which is used to determine a location of requirement (each *Shortname* has its deal in requirement number). It is recommended to use up to three characters as a *Shortname* to facilitate efficient navigation. Each requirement group contains one or more requirement or another subgroups. On each requirement group a superuser can grant or revoke privileges. So only dedicated users can view or edit content of the whole subtree.

Requirement Parameters

On each requirement group, privileged user can link a new requirement parameter definition¹⁰. It involves attaching new parameters to each requirement in a subtree. Thanks to this, there is no strict rules in requirement definition, cause each subtree can have its own list of parameters. Usually, requirements in Customers subtree consist of narrow set of parameters (regarding to production requirements). It is up to product manager or client attendance, how the customers project is configured. As was mentioned above – Requirement contains user defined Parameters.

Requirement Parameter Definition

Requirement Parameter is an instance of parameter definition (see Figure 2). It specifies a *Name*, *Description* and a *Type* of parameter. It also contains parameter *Importance* which is used to order parameters on presentation layer. Parameters can be distinguished in two categories: versioned and non-versioned. The modification of versioned parameters increases version (major version number) of requirement. Other parameters are additional parameters and have fewer content-related influences. Currently, the following *Types* of Parameters are supported:

- text, textarea – a field carrying text data, which can be in RTF format
- checkbox – a field with two logical values (true or false)
- list – a pre-defined list of values
- combobox – a pre-defined list of values shown as a single-select option field
- file – an attachment, which can be uploaded, downloaded and deleted

⁹ Group of requirements – a unit containing requirements which refer to a functionality type (e.g. GUI requirements) or subsystem (e.g. billing system)

¹⁰ Requirement parameter definition – an entity describing a type of requirement parameter

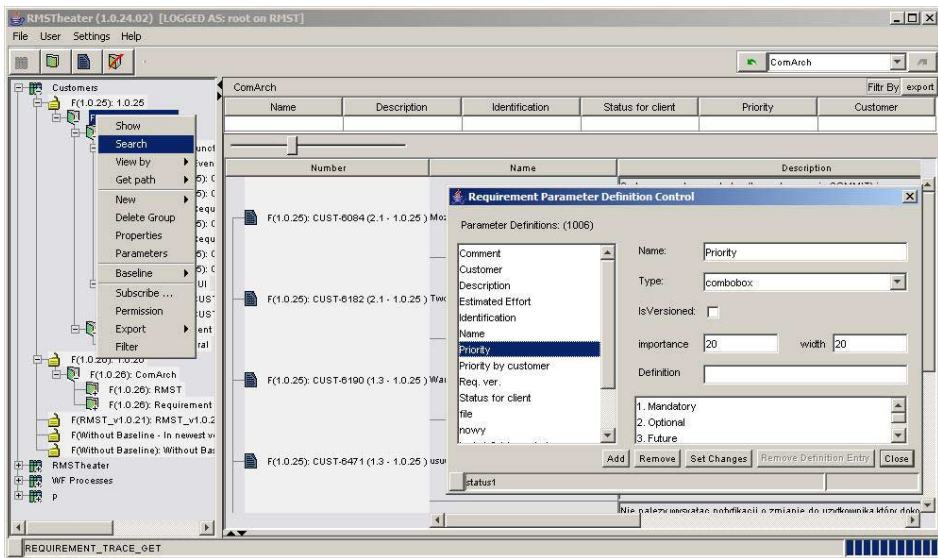


Figure 2. Requirement parameter definitions

Baselines

Baselines are major functionality of the RMSTheater. They bind a requirement to a given version of a developed software product.

Baseline has several attributes, which are:

- *Name* – a name of a baseline, should be related with product version (e.g. RMST v1.0.30), and should be unique in context of a project
- *Description* – a longer phrase performing additional information
- *Start of realisation* – a point in time, when the set of requirements binded to the given baseline is forwarded to production,
- *End of realisation* – an estimated date when all of requirements should be accomplished
- *Status of baseline* – (will be discussed below)

Baseline has two possible *Status* values: *Open* and *Closed/Frozen*. Only the product manager is able to change the value of this field. Re-opening baseline has a serious consequences and should be performed only in reasonable circumstances. A set of requirements binded to a baseline is usually a base of contract. If the modification has an impact on a scope of work it has to be agreed with business partners and should be done with respect to effects.

Requirements are added to baseline with a given version. Even though in a future someone modifies an important parameter of requirement which triggers a version increase, the content of a baseline remain unchanged. Of course, it does not mean that other users must not modify baselined requirements. They can still discuss about and adapt them to changed opportunities. So the requirements are in permanent change, like a rolling stones.

Remember, that requirement parameter modification increases of the version or history (if a parameter does not involve increase of version).

Thanks to this, project manager has a certainty that the R&D will implement a system with desired requirement versions.

2.2. Requirement Identity

One of the basic rule in requirement management is to identify requirement from the beginning to the end (when they are implemented). According this – requirement has it's unique identification number (Id). The RMSTheater system stores information about the author of a requirement and all other users which have modified it in a past. Sometimes, a person who registers requirement is not an actual author. In that case, registerer should note in a dedicated parameter *Identification* a real author. It is very important in case of any inaccuracy or doubts owing to text ambiguity.

2.3. Cooperation with External Systems

The RMSTheater is able to cooperate with external systems. Till know, it is integrated with ComArch Tytan WorkFlow and ComArch Ocean GenRap.

ComArch Tytan WorkFlow

As the name says – Tytan WorkFlow is a system organising flow of work. Suppose, that it is used as a web based communication channel between a customer and a company. Suppose than, that customer defines a new request for service or maintenance (from clients point of view application does not behave as he expected). But in a specific situation, the request is not an application error but only a suggestion. So the responsible worker in a company, should forward it to the RMSTheater as a new requirement. Such integration has a few profits – product manager has a feedback from end-users and also R&D, client has an information in which version its request will be solved.

ComArch Ocean GenRap

Ocean GenRap is a new revolutionary text processor. It has full integration with any SQL-compatible database and is able to fetch data on-line and compose a document. The RMSTheater uses this component for generating reports and listings for contracts.

2.4. System Architecture

The RMSTheater system has a three layer architecture: the customer application (Java standalone or applet), webservices (Jakarta Tomcat with Apache Axis) and jdbc-compatible database (e.g. hipersonic SQL, Oracle, MySQL,...). Hibernate is used as a persistence layer. All necessary components are open source and free for non-profit and commercial usage.

2.5. Team Works

The RMSTheater supports team work by an ability to:

- perform discussions in context of a requirement
- do asynchronous notification about changes (by email)
- communicate with a system users via an electronic mail

Each requirement has its own discussion where people can present their point of view and, if necessary, details vague meaning of requirement. The RMSTheater gives also mechanism of notifications. Concerned users can subscribe on strictly dedicated events (e.g. parameter modification, adding/removing requirement to/from baseline, ...). System will inform them by an appropriate email containing reason of notification.

2.6. Requirement Automatic Verification

The requirement analysis can be efficiently aided by simple text analysis. The main indications are listed below:

- Lines/Sentences of Text – physical lines of text as a measure of size; usually, a good quality requirement is composed of less than four sentences
- Imperatives – the existence of words and phrases that command that something must be done or provided; the number of imperatives is used as a base requirements count
- Continuances – phrases that follow an imperative and introduce the specification of requirements at a lower level, for a supplemental requirement count
- Directives – references provided to figures, tables, or notes
- Weak Phrases – clauses that are apt to cause uncertainty and leave room for multiple interpretations, measure of ambiguity (e.g. a few (how many?), fast (how fast?), emphasised (how?), ...)
- Incomplete – statements within the document that have TBD (To be Determined) or TBS (To Be Supplied), in final version their existence it is not acceptable

3. Overview

First system, which requirement management software should cooperate with is system for managing test to validate implementation of this requirements.

An advantage that RMST have over Rational RequisitePro and Borland CaliberRM is use of open source solutions that allow simple way to cooperate with existing systems.

Such important information like requirements to products that are build should not be outsourced. What is more it should be very well protected by ciphering transmissions (RMST supports that), and even the repository.

References

- [1] Olson, T. G., Reizer N. R., A Software Process Framework for the SEI Capability Maturity Model, J. W. Over (1994)
- [2] FURPS+ and RUP methodology, www-128.ibm.com/developerworks/rational/library/3975.html
- [3] Daniel M. Berry, Erik Kamsties and Michale Krieger, From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity
- [4] Magnus Penker, Hans-Erik Eriksson, Business Modeling With UML: Business Patterns at Work, *John Wiley & Sons*

- [5] Dean Leffingwell, Don Widrig, Managing Software Requirements, *Addison Wesley*
- [6] Naval Air System Command, Department of the Navy "Requirement Management Guidebook"
- [7] Software Requirements Management Working Group "Requirement Management Guidebook"
- [8] Software Requirements Management Working Group "Requirement Management Process Description"
- [9] Software Requirements Management Working Group "Requirement Management Entry, Task, Verification, Exit Charts"
- [10] Rational RequisitePro, www-3.ibm.com/software/awdtools/reqpro/
- [11] Borland CaliberRM, www.borland.com/caliber

Implementation of the Requirements Driven Quality Control Method in a Small IT Company

Stanisław SZEJKO^a, Maciej BROCHOCKI^b,
Hubert LYSKAWA^b and Wojciech E. KOZŁOWSKI^b

^a *Gdansk University of Technology, ETI Faculty,
Narutowicza 11/12, 80-952 Gdansk, Poland,
e-mail: stasz@pg.gda.pl,
<http://www.eti.pg.gda.pl>*

^b *INFO TECH sp.j., Edisona 14, 80-172 Gdansk, Poland,
e-mails: {Maciej.Brochocki, Hubert.Lyskawa, Wojciech.Kozlowski}@infotech.pl
<http://www.infotech.pl>*

Abstract. The paper describes an actual software improvement process that was carried out in a small company specialized in providing software and hardware design services in the field of real-time systems. The implemented RDQC method addresses a software project by controlling its quality on the basis of the project requirements. This is done by specifying the requirements, mapping them to quality characteristics and controlling the development process towards the demanded quality. A short presentation of the company and the pilot projects is followed by the description of the method implementation process and its results. Encouraging remarks on the usefulness and profitability of the method conclude the paper.

Introduction

INFO TECH is a small but resourceful information technology company located in Gdansk. The company is specialized in providing software and hardware design services in the field of real-time monitoring and control systems. The primary areas of competence are communication solutions for the whole variety of media and protocols, signal processing, distributed application programming, systems integration and engineering. The company is also supplying own products such as gateways, OPC servers, protocol analyzers and automated test station tools.

INFO TECH is a renowned provider of outsourced design and consulting services to foreign corporations manufacturing industrial automation systems. Typically, the software oriented projects of INFO TECH provide dedicated software to hardware platforms designed by customers. As a result, the project run is determined by the business decision model and project management model of the customer. Thus, naturally the software quality cannot be considered apart from the overall product quality assurance procedures. Requirements management must reflect business decisions of the customer in all phases of the product creation. Verification and validation steps have to be performed on the product level. INFO TECH is though

responsible for software quality assurance and the maturity of the software development process is monitored by its customers. Shared product maintenance responsibility makes both parties interested in minimizing the costs of corrective actions after the product release.

INFO TECH's autonomous position in the outsourced software development allows the company to apply the preferred development model. INFO TECH's target is to retain the CMM level not lower than 2 based on the introduced organization schemes and procedures [10]. But the company is also motivated to experiment with latest results of software quality research and focus on introducing such quality control practices, which could be directly comprehensible to the customers. An approach called Requirements Driven Quality Control was suggested by the experts from Gdansk University of Technology (GUT). The rationale of this choice was as follows:

- RDQC brings the quality characteristics directly linked to the requirements, which make them traceable not only internally, but also by the customers. This could possibly be an important advantage in INFO TECH's marketing activities.
- RDQC assumes active monitoring of the development process to retain the desired quality level.
- QFD and GQM approaches that have inspired the RDQC method are widely accepted in the industrial world.
- Good experience in earlier GUT and INFO TECH co-operation.
- Last but not least, the method can be introduced with very modest cost burden to make the first assessment of its impact. However, an extra effort was dedicated to evaluate the method and its related tools as well.

The paper describes a process of an actual implementation of the RDQC method in two INFO TECH projects and is arranged as follows: Section 1 gives a brief presentation of the pilot projects carried out using the RDQC method, Section 2 introduces the method itself, Sections 3 and 4 discuss the implementation process and its most meaningful results. Conclusions on the RDQC method and its usefulness are given at the end.

1. Selected Projects

One of the main company concerns during implementation and evaluation of the RDQC method was to perform it smoothly, without serious disturbance of the development process, and to carry it at a minimal level of additional costs that could severely harm project's business goals.

At first, an archived project was selected for post mortem analysis based on RDQC approach. The purpose of this step was mainly education rather than evaluation – to let INFO TECH team learn the method, and to let the research team from GUT recognize the development process of the company. During this phase the required measurements were selected, defined and validated. For optimization purposes they were grouped, then the supporting methods and tools (like paper forms, templates, spreadsheets or scripts) were implemented. This phase of the RDQC method comprehension and evaluation was done without active cooperation of the project development team,

involving only the project manager and two quality engineers. The results were later evaluated and discussed with the company management.

Following this, an upcoming software development project was selected to be used as RDQC pilot study, conducted as joint effort of INFO TECH and GUT. The selection criteria were the low level of business risk of the project and well-maintained technology applied in the project, with software reuse. The subject of the project scope was to develop a communication subsystem for a dedicated embedded device. This complex project was divided into two subprojects. The first subproject (called *Alpha project*) was aimed to develop the communication interface for an embedded device used in remote monitoring of the power distribution network. The other subproject (*Beta project*) was aimed to develop a configuration toolset running on PC platform or handheld and used for the device from Alpha project. While Alpha project was strongly customer-driven, Beta project could be seen as more autonomous.

In general, both projects followed the waterfall model. However, the company solutions reduce some activities, like modeling or interface prototyping while more emphasis on reusability of company-developed components and testing is put.

During the RDQC method implementation its subjects and activities were incorporated into the quality assurance plan of the upcoming Alpha and Beta projects. The development team was trained for the use of the method and the new RDQC tasks were assigned and performed within Alpha and Beta projects' scope. Having in mind the results of the post mortem analysis and based on measurements from the ongoing projects, selected corrective and quality improving actions were imposed. After the projects completion the results were evaluated and discussed within the development team and the company management.

2. A Method of Requirements Driven Quality Control

The primary aim of the RDQC method is to link software quality improvement throughout the development process with the customer needs. This is done by selecting the most important quality characteristics¹ due to the identified user requirements and needs, and controlling them during the development stages. Putting focus on the selected quality characteristics should also result in lowering the cost of software quality activities – in comparison with conforming to quality standards and mature development models [5].

Thus the proposed method consists of three steps: requirements specification, selecting the demanded quality characteristics on the basis of this specification, and controlling quality of the software system under development. To lower the risk of the method practical usage the steps are supported with well-known and approved approaches.

2.1. Requirements Specification

After the requirements are identified they are grouped according to several predefined categories. These categories have been established following “the widest” EWICS stan-

¹ *Subcharacteristics* according to ISO/IEE 9126 Standard [4], and *criteria* according to the Factors–Criteria–Metrics model [2]

dard for safety critical systems [11] and include: *system objectives, functional and non-functional requirements* (the latter include those directly addressing quality), *target environment requirements and limitations*, as well as *development requirements*.

The specification should include and prioritize all the requirements and not be limited to those expressed explicitly by the customer. In order to properly evaluate mutual significance of the requirements affinity diagrams can be applied to form their hierarchies so the weights of superordinated requirements can be calculated on the basis of weights of the subordinated ones.

These concepts were implemented in a specification-supporting tool that allows also for grouping projects according to system classes so that core requirements for a given class can be prompted [1].

2.2. Selecting the Demanded Quality Characteristics

The step aims at converting the specified requirements to the demanded software quality. The idea is to assume a generic model of quality and to map the requirements to it thus obtaining the most significant quality characteristics.

Hence, the step consists in:

1. Selecting a quality model – the RDQC version applied in INFO TECH used a model that had been developed during the SOJO (QESA) Project [7], [8]. A new RDQC supporting tool under development is going to introduce also a model that conforms to the ISO/IEC 9126 standard [4].
2. Weighting the requirements – the essential problems are: what factors should be taken into consideration, how should they be valued and by whom. RDQC applications bring two possibilities: either factors that follow House of Quality matrix of the Quality Function Deployment method [6], [9], like rate of importance, company current situation and plan, rate of improvement, are defined by an interdisciplinary team, or experience, knowledge and range of responsibility of persons defining requirements are taken into consideration (and as a heuristic algorithm proposed in [1] requires better validation during RDQC implementation in INFO TECH the related coefficients were set at some discretion).
3. Mapping requirements to quality characteristics and selecting the most significant ones. Setting a correlation between each requirement and every characteristic each time requires answering the question *about the degree that a given characteristic impacts on satisfying the requirement*. We use the scale suggested in HoQ/QFD by setting strong (9), some (3), weak (1) or no correlation between quality characteristics and requirements. These make basis for calculating direct and indirect weights of characteristics. Prototype tools support the step and a Pareto-Lorenz diagram of the highly demanded quality characteristics can be observed.

Alternatively, the selection can be done following the analytic hierarchy process (AHP, [9], [12]) of making comparisons between pairs of characteristics under the requirements criteria.

2.3. Controlling the System Quality

We focus on measuring and improving the set of selected characteristics throughout the development process. The scheme follows the Goal/Question/Metric (GQM, [13]) approach: products are defined for each development phase and questions and metrics are set to evaluate every required characteristic as shown in Table 1.

Table 1. Illustration of the CQM approach

	Analysis (Modeling)	Design	Coding & testing
Characteristic	Functional completeness – a degree to which required functions are provided		
Question	Do the system representations include all the required functions?		
Metric	Percentage of functional requirements covered by the functional test specification	Percentage of functional requirements covered by the class model	Percentage of functional requirements fully implemented
Products (deliverables)	Requirements specification, Functional test specification	Requirements specification, Class model	Requirements specification, Code

Of course, in case the measurements show that the characteristics are unsatisfactory the corrective actions shall be taken: the development stage may be repeated or improved. And as characteristics make goals of this step we call it Characteristic-Quality-Metric, CQM. A prototype tool supporting this phase and a prototype database of predefined questions and metrics have been developed [1].

Some wider description of the RDQC method can be found in [8], [14], [15].

3. Implementation of the RDQC Method

Implementation of the RDQC method was very carefully planned. Extensive plan of measurements included measurements of the project deliverables and groups of measurements for evaluation of the RDQC method itself. Draft schedule of actions contained many mandatory and optional tasks and the responsibility assignments.

Project team members were involved in metrics evaluation and participated in presentations devoted to the RDQC method. Most of the work was done by quality engineer under a supervision of the project manager and an expert from GUT. Such assignment of responsibilities let us discover some undesired situations (e.g. project manager could influence quality assurance evaluations) and gave an opportunity to determine a better organizational solution for future implementations.

Table 2. List of tasks with roles assignments

Task	Project role
<ul style="list-style-type: none">• preparing a complete list of requirements• grouping the requirements	system analyst

• weighting the requirements	system analyst acting as customer representative
• determining correlation between requirements and quality characteristics	independent experts
• mapping reference project deliverables to the actual set of project deliverables in the company	quality engineer
• definition of questions and metrics	
• presentation showing how the project will be assessed and encouraging to apply the method	project team, quality assurance unit
• development activities	project team members
• measurements and evaluations	quality engineer, project team members
• preparing summaries and reports	quality engineer
• deciding about quality improvements	project manager
• post mortem presentation with the summary of quality control results	project team, quality assurance unit

3.1. Requirements Specification

As mentioned before this was assembled by the system analyst. Requirements were divided into the following categories: physical layer, configuration parameters, protocol objects and functions, performance, mapping conventions and engineering support (the two latter categories dealt mainly with Beta project). Requirements specification for the products was supplemented with company requirements concerning the development process.

3.2. Analysis of the Quality Characteristics Significance

Analysis of the important quality characteristics was performed for both Alpha and Beta projects. The priorities were given by three independent experts on the very early stage and the average values were calculated. Correlation coefficients were set following the HoQ/QFD pattern what led to indirect values of the characteristics. The results of the analysis for 12 most significant characteristics are given in Table 3.

Table 3. Significance of quality characteristics for the analyzed Alpha and Beta subprojects

Quality characteristic	Alpha [%]	Beta [%]
Functional completeness	14,14	12,72
Acceptance	13,42	10,68
Ease of use	6,54	17,36
Complexity	6,54	3,44
Productivity	6,43	5,66
Execution efficiency	5,82	1,86
Adequacy	5,55	4,92

Integrity	5,49	5,11
Configurability	5,16	6,78
Modifiability	2,83	5,01
Understandability	2,55	5,29
Interaction performance	1,39	7,80

We can see that in Alpha project *functional completeness* and *acceptance* have got much higher ratings than other characteristics, so they were next selected to be controlled in the quality control step.

The same two features were also seen as highly significant in the Beta project, however the highest rating was calculated for the *ease of use* – and those three characteristics were chosen for the quality control phase.

Results of analysis of important quality characteristics confirm that fulfilling the contract agreements (functional completeness) and acceptance of delivered solution by the customer are the most important for the INFO TECH company. Similar results for both projects can be caused by the strong relation between the projects and their placement in a very specific domain. Assuring the ease of use seems very important for the end-users of the Beta configuration tool.

Experts' evaluations of the correlation coefficients were not unison, which might not guarantee the repeatability of the analysis. According to our experience, this was implied by different understanding of the defined model characteristics. Simple procedure, e.g.:

- read definition of a quality characteristic carefully,
- determine correlation between assuring this quality characteristic and fulfilling the following requirements,
- repeat previous steps for all the characteristics,

can help as it allows keeping the correct and consistent definition of each quality characteristic throughout the whole assessment process.

3.3. Quality Control

On this basis the quality engineer has created definitions of questions and metrics. This was done with the help of questions and metrics predefined in a RDQC/CQM supporting database [1]. As the database assumes the object-oriented approach within the waterfall model, mapping of its project deliverables to the actual set of INFO TECH project deliverables was necessary (it occurred that this project process did not include class modeling, data dictionary as well as user interface prototyping). On the other hand the predefined database did not contain many metrics referring to test specifications and test reports.

Systematic approach was used to select and define new questions. At first, all aspects having impact on a given quality characteristic were identified. For example for functional completeness the following aspects were identified: presence of all required functions, exceptions handling, description of sequence of events, description of processed data, completeness and consistency of project deliverables, support of all system interfaces, support for all groups of users. Next, a question for each aspect was

defined and assigned to the quality characteristic in each phase. For example for presence of all required functions the following question was defined: “Do the system representations include all the required functions?”. The main goal of this systematic approach was to facilitate the interpretation of the results, however this approach was not directly showing the areas of required improvement. That was why the metrics were also grouped based on the areas of responsibility of the project roles. A total number of seventy five metrics was defined. All the measurements were performed manually by the quality engineer.

Summarized results of the first measurements in Alpha project are presented in Table 4. Similar results were obtained in Beta project. Project manager had a comprehensive view on the quality of both projects.

Table 4. Results of the first measurements for Alpha project (“—” indicates an absence of metrics verifying a certain aspect of a quality characteristic)

Alpha Project	Question	Spec.	Design	Code, Test
Functional completeness	Do the system representations include all the required functions?	74%	66%	95%
	Do the system representations consider exceptional situations handling?	34%	6%	78%
	Do the system representations describe sequences of events?	96%	—	—
	Do the system representations describe processed data?	50%	15%	94%
	Are the system representations complete and consistent?	88%	78%	68%
	Are all the system interfaces considered?	50%	0%	—
	Overall results	65%	33%	84%
Acceptance	Is the system useful?	95%	—	74%
	Does the protocol implementation conform to the standard?	81%	—	90%
	Are the system acceptance criteria met?	75%	63%	67%
	Overall results	83%	63%	77%

Both projects showed worse quality results in the design phase. Most likely this was because the reuse-based design was thought not to require so much attention, there was less effort put on technical inspections and no demand of providing detailed design level documentation to the customer (design documentation was prepared only for INFO TECH internal use). Deliverables and metrics for Beta project assumed business modeling and higher focus on end-user tasks, which led to worse results.

Two cycles of quality control were done and two series of measurements were performed.

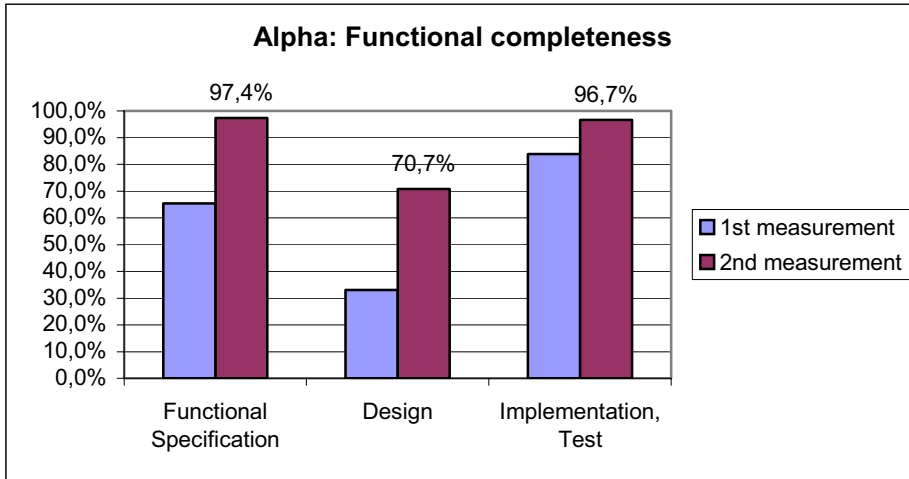


Figure 1. Results for functional completeness in Alpha project

Results were also gathered according to the project roles, as presented in Figure 2.

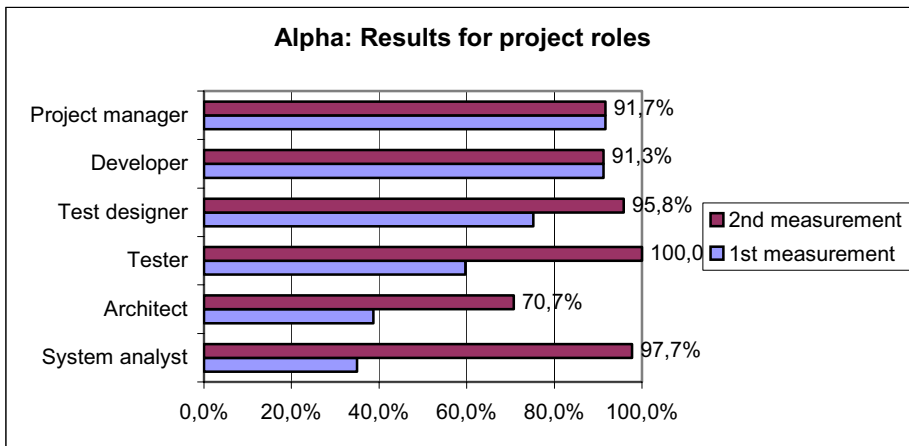


Figure 2. Results for project roles in Alpha project

Both series of measurements were performed on almost the same project deliverables. Much higher results of the second measurements for Alpha project were caused by taking into consideration explanations of the project team members. Those explanations reflected implicit decisions made. Scope of the documentation and test cases were limited, because of reuse and usage of external specifications. In case of Beta project the test specification was updated and we could observe high quality improvement in this area.

4. Evaluations and Assessments

Project team members filled a questionnaire assessing the RDQC method and the results of the implementation. Results of analysis of important quality characteristics were considered more reliable, but still the results of measurements were also found very useful. Most of the project team members agreed that it would be worth to continue the usage of the RDQC method. Project manager of the assessed projects confirmed that the RDQC method showed its usefulness giving a higher control over the projects.

Implementation of the RDQC method summed to 11% of the total project effort while in literature discussed in [3] the costs of achieving quality and removing failures detected prior to shipment are over 25% for companies at CMM level 3, and higher for lower levels of maturity.

Cost sharing is presented in Figures 3 and 4 below.

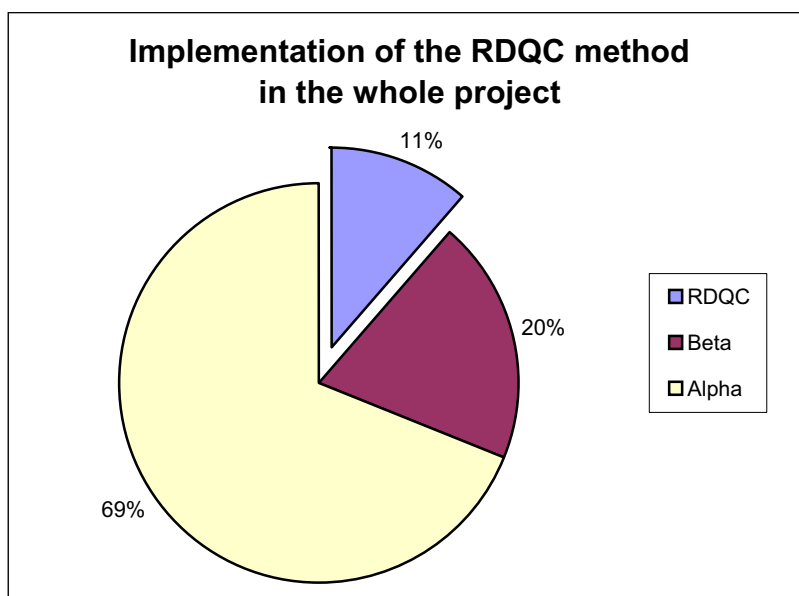


Figure 3. Cost of the RDQC method in the whole project

Efforts of the future implementations should be lower if a better database of predefined questions and metrics is available. Definition of additional metrics will not be necessary, because its goal was to assess the RDQC method. We have to remember that an initial implementation of any method has higher costs. A positive feedback to the introduced comprehensive quality control measures from the project team members was also noticed despite of an extra effort.

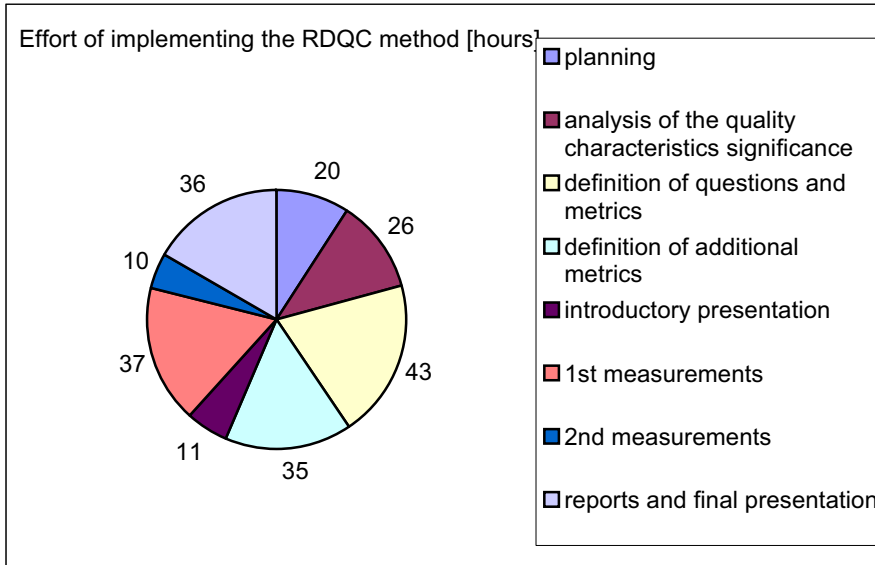


Figure 4. Cost of particular tasks

5. Conclusions

The RDQC method showed its usefulness by giving a comprehensive view on the quality of the assessed projects and pointing out the areas of required improvement. It is worth to emphasize the significance of post mortem analysis and careful planning of the implementation. The implementation of the RDQC method helped to detect many practical, organizational and psychological issues. Tools supporting the method were used and their improvements were proposed.

The method proved its applicability to a company-specific software development process. The relatively low cost overhead confirmed the original expectations and justified the applied measures to reduce the risk of higher costs of future corrective maintenance tasks. Even more benefits of the method can be expected in projects with a lower degree of software reuse.

Future implementations of the RDQC method should benefit from the gained experience and the solutions established for the RDQC management – planning, setting team roles, process coordination and documentation. Costs of future implementations should also be decreased due to the developed tools and the created CQM database.

Current works concentrate on defining a development process with RDQC method applied and on improvement of tools supporting the method. It is planned to introduce the internal pilot results to the customers and coordinate with them the future measures of software quality control in joint projects. This should allow for better evaluation of the concept of controlling quality attributes selected on the basis of system requirements.

References

- [1] Barylski M., Brochocki M., Kolosinska K., Olszewski M., Pardo K., Reclaf M., Szczerkowski M., Tymcio R.: Reports and master thesis in the field (in Polish). *Software Engineering Dept., ETI Faculty, Gdansk University of Technology*, (1999-2004)
- [2] Fenton N.E.: *Software Metrics. A Rigorous Approach*. Chapman & Hall, (1993)
- [3] Houston D., Keats J.B.: *Cost of Software Quality: A Means of Promoting Software Process Improvement*. Arizona State University, SDM Group <http://www.eas.asu.edu/~sdm/papers/cosq.htm>
- [4] ISO 9126: The Standard of Reference. <http://www.cse.dcu.ie/essiscope/sm2/9126ref.html>
- [5] Jeletic K., Pajerski R., Brown C.: *Software Process Improvement Guidebook*. Software Engineering Laboratory, NASA-GB-001-95 Report (1996)
- [6] King B.: *Better Designs in Half Time, GOAL/QPC* (1989)
- [7] Krawczyk H., Sikorski M., Szejko S., Wiszniewski B.: A Tool for Quality Evaluation of Parallel and Distributed Software Applications. *3rd International Conf. on Parallel Processing & Applied Mathematics*, (1999) 413-426
- [8] Krawczyk H., Sikorski M., Szejko S., Wiszniewski B.: Evaluating of Software Quality Characteristics Significance (in Polish). *II National Conf. on Software Engineering, KKIO'2000, Zakopane, Poland (Oct 2000)* 179-186
- [9] Madu Ch.N.: *House of Quality in a Minute*. Chi Publishers, (2000)
- [10] Orci T., Laryd A.: *CMM for Small Organisations. Level 2*. UMINF-00.20. (2000)
- [11] Redmill F.J. (ed): *Dependability of Critical Computer Systems*, vol. 1. Guidelines produced by European Workshop on Industrial Computer Systems, *Elsevier Applied Science*, (1988)
- [12] Saaty T.L.: *The Analytic Hierarchy Process*. McGraw-Hill, New York (1980)
- [13] Solingen R., Berghout E.: *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill (1999)
- [14] Szejko S.: Requirements Driven Quality Control. Proc. XXVI Computer Software and Applications Conference COMPSAC 2002, Oxford, UK, 2002, *IEEE Computer Society Press*, (2002) 125-130
- [15] Szejko S.: RDQC – A Method of Requirements Driven Quality Control (in Polish). *VI National Conf. on Software Engineering, KKIO'2004, Gdansk, Poland, (Oct 2004)*

System Definition and Implementation Plan Driven by non-Linear Quality Function Deployment Model

Tomasz NOWAK ^a and Mirosław GŁOWACKI ^b

^a *ABB Corporate Research,
ul. Starowiślna 13A, 31-038 Krakow, Poland
e-mail: Tomasz.Nowak@pl.abb.com
<http://www.abb.com/plcrc>*

^b *AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: glowacki@metal.agh.edu.pl*

Abstract. In this paper the systematic approach for optimal definition of system functionality and implementation plan has been described. Non-linear decision model, with goal function aiming to maximize the level of customer satisfaction, and considering budget constraints as well as risk parameters is given and explained. The practical application of the proposed methodology is presented and discussed. Definition of the system architecture and implementation plan for computer-based collaboration environment supporting communication within the cross-bordered manufacturing company is illustrated.

Introduction

In recent years, the software engineering community has paid considerable attention to the right definition of system architecture, which is based on customer requirements. With the increasing complexity of project design, shorter development time, and tight budget constrains the need for clear system specification, which would fulfill the functionality requested by users, is essential.

However, different perspectives for system development represented by project stakeholders make the product specification difficult to be defined, and very often the customer needs are not accurately fulfilled. In order to specify a comprehensive set of system characteristics some systematic approaches may be used. Among several methods for system specification, Quality Function Deployment (QFD) should be highlighted.

In general, the Quality Function Deployment, which was developed in Japan by Yoji Akao in 1972, provides a systematic methodology for translating the Customer Requirements (CRs), through the stages of product planning, engineering and manufacturing, into the complete definitions of the product such as Engineering Characteristics (ECs), process plans and process parameters, which finally meet customer demands, [1].

QFD grew out of the activities running at Toyota and shipyard industry, and it basically traced the handling of customer-demanded quality attributes by the product

design, component selection and process specification and control. The QFD method was originally referring to manufactured goods, however, the basic philosophy has also been utilized with common software quality considerations to create a software requirements management model.

In this paper, novel approach for application of QFD method into system architecture design and specification of deployment plan has been presented. Commonly used software QFD models have been extended about new decision variables, such as: needs expressed by different user groups, probabilistic distribution of project cost, and risk level accepted by decision maker. Model proposed by authors maximizes the satisfaction of users having different system requirements and allows decision maker considering non-linear budget constraints and undertaking different risk strategies (optimistic, realistic or pessimistic). The model was validated during the project that aimed to develop and implement the computer-supported collaboration system for global manufacturing company.

1. Related Work

In general, the core idea of the QFD philosophy relays on translating a set of customer requirements, drawn upon market search and benchmarking data, into appropriate number of prioritized engineering characteristics to be met by new product/ system design. The most recognized form of QFD is “House of Quality” matrix, which usually consists of six major components [14], [2], [8]:

1. Customer Requirements, CRs – a structured list of needs derived from customer statements (WHATs);
2. Engineering Characteristics, ECs – a structured list of relevant and measurable product technical definitions (HOWs);
3. Planning Matrix – includes relative importance of customer requirements and company vs. competitor performance in meeting these requirements;
4. Interrelationship Matrix – illustrates the QFD team’s perception of relationships between CRs and ECs;
5. Technical correlation (Roof) matrix – used to identify which ECs support (or impede) each other;
6. Technical priorities, benchmarks and targets – used to highlight technical priorities assigned to ECs and plan target values for each EC, which are linked back to the demands of the customer. It is the main outcome of the QFD tool.

The scheme of the “House of Quality” is shown in Figure 1.

QFD is firmly formalized methodology, but the expression of customer requirements and their rating often inherits some forms of imprecision, vagueness, ambiguity, uncertainty and subjective assessment.

Software is an intangible product, and several attempts have been made in order to cope with the difficulties in carrying out the Quality Function Deployment processes. The most basic issue was to establish some universal measures for customer needs. As a starting point, the foundation of ISO 9126 standard [15] proposed six primary product categories: Functionality (suitability, accurateness, interoperability, compliance, security); Reliability (maturity, fault tolerance, recoverability); Usability (understandability, learnability, operability); Efficiency (time and resource behavior); Main-

tainability (analyzability, changeability, stability, testability), and Portability (adaptability, installability, conformance, replacability). Similarly, the IEEE 830:1998 standard [7] defined following software requirements, such as: Functional, Performance, Security, Maintainability, Reliability, Availability, Database, Documentation, and Additional requirements.

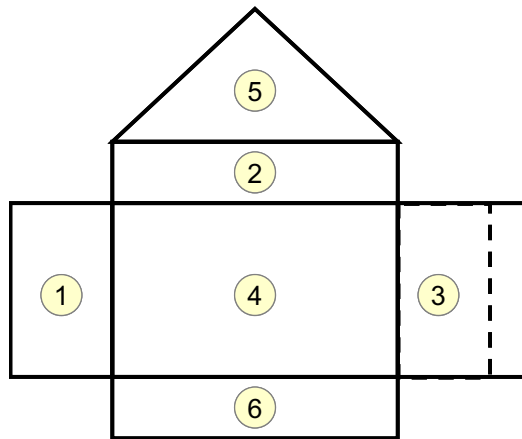


Figure 1. Graphical representation of the *House of Quality*

Beside the problems with establishing the universal measures for customer needs, other essential QFD topics consider: application of ambiguous or fuzzy (linguistic) expressions for judgments of CRs importance, and systematic approaches to determine priorities of ECs. For example Masud & Dean [9] propose an approach for prioritizing ECs, where weights of the CRs and relationships between CRs and ECs are represented by triangular fuzzy numbers, which are used to calculate the fuzzy priorities, converted finally into real numbers. Park & Kim [12] also present a decision model for prioritizing ECs, but they use crisp data. Their procedure uses Analytical Hierarchy Process to define the relative importance of the CRs, assigns weights to the relationships between CRs and ECs, integrates roof of the House of Quality into the decision model, and selects the set of ECs which has the biggest influence on the customer satisfaction.

But very few researches have attempted to find optimum EC targets, which is an important issue during application of QFD into software projects. Due to budget and time constraints all user needs cannot be fully met, therefore some trade-offs must be made. Traditional approaches for setting the expected targets for CRs are usually subjective [5], offering feasible solutions rather than optimal ones. Moskowitz & Kim [10] present an approach for determining targets, which is based upon mathematical programming – the level of satisfaction produced by an EC value per CR is expressed as a function. However, this formulation did not take into consideration the design costs. Dawson & Askin [3] propose the use of a nonlinear mathematical program for determining optimum ECs, taking into account costs and development time constraints. But, as stated by these researches, more work has to be done to address the ability to account for dependence among ECs and to assure the robustness of the approximated

value function utilization. Correlations between different ECs have been included in mathematical model proposed by Fung et al. [6]. These authors assume also that costs for ECs attainment can be represent in fuzzy sense.

These mentioned above works are useful, but they all lack some aspects, important especially when applying QFD methodology into software projects.

First of all, presented approaches assume that all software users have quite similar expectations for the system, while in reality, there can be different groups of customers having very specific (and even contradictory) needs. This situation concerns especially real-time communication systems, aimed to enhance computer-supported collaborative work (CSCW), which are used by variety of stakeholders having different needs, experience and IT knowledge.

Secondly, the cost for attainment of given EC is not a crisp data and should be treated as a probabilistic number – with its expectation and standard deviation. Because there is the likelihood, that some tasks will be implemented faster (cheaper) or longer (more expensive) then expected. The EC cost ought to be estimated using Normal or Weibull distributions, rather than by applying fuzzy numbers, as suggested by some authors. It is also worth mentioning, that cost of specific task may be generally expressed as a sum of two components: (i) fixed- and (ii) variable with task attainment. The first component describes all the rigid costs, which have to be covered on the task onset – purchasing of needed software and hardware, training courses, etc. The second component reflects labor and material costs mainly.

The implementation of the above-mentioned considerations and assumptions into the decision model is presented in next chapter.

2. Mathematical Model

As stated before, QFD translates the customer requirements (CRs) into engineering characteristics (ECs). Technical priority describes the importance of given EC perceived by customer. Mathematically, it can be expressed as follows:

$$r_j = \sum_{i=1}^m (w_i \cdot z_{ij}) \quad \text{for } j = 1, 2, \dots, n \quad (1)$$

where: r_j = importance (technical priority) for j^{th} EC; z_{ij} = correlation between i^{th} CR and j^{th} EC (contribution of j^{th} EC towards i^{th} CR attainment); and w_i = customer perception of i^{th} CR importance (weight of i^{th} CR).

In real cases the system must meet different ECs, therefore also a set of technical priorities have to be analyzed. Let the decision variable x_j be the actual attainment for the j^{th} EC, $0 \leq x_j \leq 1$, and $x_j = 1$ when the target is reached. Then formulation of a QFD planning problem describing maximization of customer satisfaction F can be written as below:

$$\max F = \sum_{j=1}^n r_j x_j \quad (2)$$

And let the decision maker consider two groups (“A” and “B”) of users representing different perceptions of i^{th} CR. Then expression for equivalent (weighted)

customer satisfaction F' can be given as linear combination of two customer satisfaction components:

$$\max F' = \max(F^A + F^B) = \beta \sum_{j=1}^n r_j^A x_j + (1 - \beta) \sum_{j=1}^n r_j^B x_j \quad (3)$$

where: F^A, F^B = satisfaction of „A” and „B” group, respectively; r_j^A, r_j^B = technical priority for j^{th} EC represented by „A” and „B” group, respectively; x_j = actual attainment for j^{th} EC; and β = coefficient of group importance – defined by decision maker, $0 \leq \beta \leq 1$, ($\beta = 0$ if requirements of group “A” are not taken into account at all, $\beta = 1$ if only requirements of group “A” are considered).

In order to include the roof of the “House of Quality” into the decision model the “actual” and “planned” attainments for given ECs should be differentiated. If the planned attainment for j^{th} EC is noted by y_j then the correlation between x_j and y_j can be formulated as follows:

$$x_j = y_j + \sum_{k \neq j}^n t_{kj} y_k \quad \text{for } j=1,2,\dots,n \quad (4)$$

where: y_j = planned attainment; x_j = actual (archived) attainment for j^{th} EC; and t_{kj} = correlation between k^{th} and j^{th} ECs.

The correlation t_{kj} between the k^{th} and j^{th} EC can be interpreted as an incremental change in the attainment for the j^{th} EC when the attainment for the k^{th} EC is increased by one unit. In other words, Eq. 4 describes positive or negative relations between ECs, (e.g.: high hardware performance drives to fast processing of user request – positive correlation, but it has negative impact on project cost).

When considering economic constraints – it must be stated that planned cost C for the realization of project tasks must fit into a given budget, B :

$$C = \sum_{j=1}^n C_j = \sum_{j=1}^n \Phi_j^{-1}(\gamma_j, c_j, \sigma_j) \leq B \quad (5)$$

where: C_j = planned cost of j^{th} EC attainment; Φ^{-1} = inverse function of cost probability; γ_j = confidence level for j^{th} task, $0 \leq \gamma_j \leq 1$, (e.g. $\gamma = 0,20$ if cost is estimated with high risk); c_j = expected (the most probable) value of j^{th} EC cost; and σ_j = standard deviation of j^{th} EC cost.

The relationships between parameters appearing in Eq. (5) are graphically presented in Figure 2.

As noted in previous section, most probable cost c_j of given EC can be expressed as sum of two components:

$$c_j = c_j^F + c_j^V \cdot y_j \quad (6)$$

where: c_j^F = cost component independent on attainment of j^{th} EC (fixed cost); and c_j^V = variable cost component dependent on task attainment (e.g. related to labor hours). The

relationship between the expected cost of j^{th} EC and the task attainment is schematically shown in Figure 3.

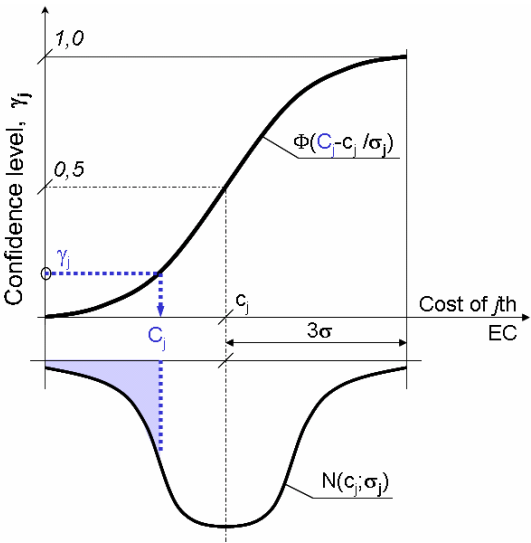


Figure 2. Normal distribution of the task cost

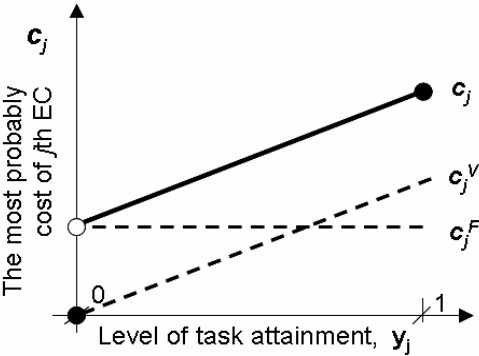


Figure 3. Cost components vs. level of task attainment

Summarizing, the proposed optimization model maximizes the goal function F' describing the linear combination of customer satisfaction for two groups of users, with non-linear financial constraints. The decision parameters are: available budget for the project, B , expected cost c_j of fully accomplished task, risk accepted by decision maker (confidence level, γ), and coefficient of group importance, β . As a result, the model provides values of planned attainments y_j for given ECs, which is in fact the basic information for decision maker – how much money (effort) should be allocated on given project task, and what level of user satisfaction will be achieved by that.

3. Practical Application of the Model

The decision model described in previous chapter was validated during realization of the real software project. The goal of this initiative was define and implement the computer-supported collaborative environment for global manufacturing company. It was observed, that widely distributed factory units suffer due to poor communication, what drives to longer and more expansive product development processes. It was expected, that powerful and attractive collaboration tool will overcome the technical and psychological communication barriers.

Project was conducted in five main phases. First, the user requirements were gathered and analyzed. Next, these customer needs were transferred into technical characteristics of the system. Subsequently, the modules of the system and optimal implementation plan were defined using proposed non-linear QFD model. Finally, system was introduced using two-step approach.

3.1. Capturing Customer Requirements

Customer requirements for collaboration system where gathered using electronic questionnaires. Authors sent out the survey files to 24 production units in 10 countries. All reviewed factories belonged to one international company, however they represented independent, and thus different, business organizations.

The electronic questionnaire contained about 40 detailed questions, which were divided into 3 main groups related to: business area (kind of business the factory is doing, main products, markets, clients, suppliers, etc.); collaborative engineering (project realization process, design reviews, CAD data exchange); and data management (data storage, accessing and transferring). According to the QFD practice – questions included in survey were stated in rather general form, since customers would not be familiar with specific techniques and solutions for computer-based collaboration systems. In order to collect all needed information also phone conferences and in-place meetings had to be organized in some cases.

3.2. Analysis of Customer Requirements

Based on the gathered data, it was possible to characterize requirements for collaboration environment, Table 1.

Table 1. Customer requirements

Data Visualization	
CR1	Big number of 3D/2D/office formats supported
CR2	Graphical visualization and model measuring
Data Modification	
CR3	Ability for model modification (e.g. radius change)
CR4	Applying of comments, notes, remarks into doc.
Design Review	
CR5	Easy session set-up
CR6	Firewall crossing
CR7	Efficiency (fast data transfer)
CR8	High security

<i>Data Management</i>	
CR9	Engineering data (storage, accessing and searching)
CR10	Project data (task and people management)
<i>Usage</i>	
CR11	Client perspective (easy to learn and use, powerful, context help, etc.)
CR12	Administrator perspective (installation, task automation, back-up, etc.)

The results of survey lead to conclusion that reviewed factories fit into two main groups: “Producers”- and “Consumers” of information, having two different need profiles, what is clearly reflected in system requirements, Table 2.

Table 2. Customer requirements importance (%)

	Producers	Consumers
CR1	11,5	5,0
CR2	11,5	5,0
CR3	3,8	1,7
CR4	11,5	15,0
CR5	3,8	15,0
CR6	11,5	15,0
CR7	11,5	5,0
CR8	3,8	15,0
CR9	11,7	5,0
CR10	11,7	1,7
CR11	3,8	15,0
CR12	3,8	1,6
total	100,0	100,0

Both: Producers and Consumers rate Design Review functionality (CR5-CR8) as the most important one, however Producers seriously consider also Visualization (CR1-CR2) and Data Management (CR9-CR10), while Consumers put their accent on easy Usage (CR11).

3.3. Mapping of Customer Needs into Engineering Characteristics

In order to translate the customer wishes into system specification a subset of IEEE 830:1998 recommendations was used, covering the primary software characteristics, such as: Functionality (outline of what the product will do for the users); Performance (speed or duration of product use); Security (steps taken to prevent improper or unauthorized use); Maintainability (ability for product to be changed) and Database (requirements for managing, storing, retrieving, and securing data from use). Following these recommendations QFD’s team mapped CRs into ECs, as presented in Table 3.

Table 3. System engineering characteristics

Functionality	EC1	3D/2D graphical kernel
	EC2	Office document processing module
	EC3	CAD/CAM/CAE translators
	EC4	E-mail server
	EC5	Session scheduling system
	EC6	Proxy server / firewall crossing
	EC7	Internet browser technology
Performance	EC8	Streaming technology
	EC9	Data compression
	EC10	Geometry simplification (e.g. VRML)
	EC11	Client-server architecture
Security	EC12	Multilevel access and password system
	EC13	SSL technology
Data Management	EC14	Database engine
	EC15	Workflow
Maintainability	EC16	Task automation: API, Open Interface, etc
	ER17	Back-up and recovery system

Next, the correlations between CRs and ECs were defined using 1-3-9 scale, Table 4:

Table 4. Interrelationship Matrix

CRs	ECs																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	3	9														
2	9	3															
3	9	9															
4	3	3															
5				3	9												
6				3		9											
7							9	9	1	3							
8											9	9					
9													9	1			
10													9	3			
11							9								3		
12							1			3						9	3

In order to fill-in the roof of the House of Quality it was necessary to describe the correlations between different ECs. This matrix is not shown in this paper (for details see [11]), however it must be highlighted that there are some obvious relationships between system characteristics (e.g. between EC7 and EC11).

3.4. Numerical Calculations of Optimal ECs Attainments

The planned attainments for all ECs were derived by implementing the mathematical decision model described in Chapter 3 in numerical calculation package. In this case Generalized Reduced Gradient (GRG2) method for non-linear calculations and Simplex algorithm for linear models [4] were used. In order to run the analysis, the relevant decision parameters: confidence level γ , coefficient of group importance β and relevant cost components c_j^F, c_j^V had to be assigned. Calculations were conducted for combination of three risk strategies: optimistic ($\gamma = 0,2$), normal ($\gamma = 0,5$), and pessimistic ($\gamma = 0,95$); and different coefficients of group importance: highlighting Producers needs ($\beta = 1$), expressing Consumers requirements ($\beta = 0$) and weighted equally ($\beta = 0,5$). Cost components (e.g. labor costs) were estimated using the mean values on the market.

As a calculation result – the level of client’s satisfaction versus project budget for different configurations was found. For example, Figure 4 presents the curve of customer satisfaction for case, in which both decision parameters are set to mean values ($\gamma = 0,5$; and $\beta = 0,5$). By error bars also the optimistic case ($\gamma = 0,2$) and pessimistic case ($\gamma = 0,95$) are considered.

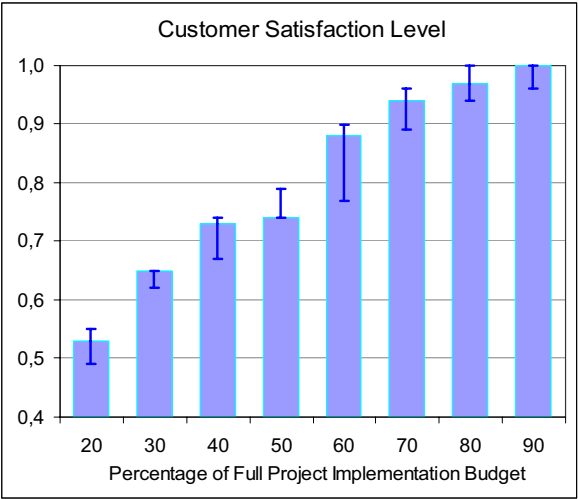


Figure 4. Customer satisfaction level vs. percentage of the project budget

From this graph it can be concluded that setting optimally attainments for ECs is possible to fulfill about 74% of customer needs by spending only 40% of maximal budget. Table 5 presents values of EC attainments for this case.

Table 5. Optimal ECs attainments for 40% of project budget

	ECs											
	2	4	5	6	7	8	10	12	13	14	15	17
y_j	1.	.8	.9	1.	.7	.7	1.	1.	1.	.6	.8	.9

Figure 4 informs also that for ensuring the customer satisfaction on level of 80% – even in pessimistic cases, it is required to spend 70% of maximal budget. In the case the project goes quite normal ($\gamma = 0,5$), these 70% of total budget will results in as much as 94% of user satisfaction, Table 6.

Table 6. Optimal ECs attainments for 70% of project budget

		ECs													
		1	2	3	4	5	6	7	8	10	12	13	14	15	17
y_j	1.	1.	.8	.8	.9	1.	.7	.7	1.	1.	1.	.7	.8	.9	

3.5. System Implementation

Following the results of analysis, it was proposed, that for this specific industry case, the collaboration system can be optimally implemented in two main project steps: up to reaching 74% of satisfaction level (what corresponds to 40% of max. project costs) and up to getting 94% of user satisfaction (70% of max. project costs). The decision maker can optionally consider stopping the project after the first step, if the level of achieved customers' satisfaction is accepted. It was also found that, there is no economic sense to spend more than 70% of max. budget. The increase of customer satisfaction, if remaining functionality is implemented, is not so big.

4. Conclusions

The presented approach for applying the non-linear QFD model in collaboration environment design allows for multi-aspect optimization of customer satisfaction. Different customer perspectives for system requirements, the risk level accepted by decision maker and budget constraints are considered in the algorithm. Application of the described decision model results in optimal selection of all system characteristics to be implemented on given project stage.

Proposed methodology was successfully verified during the implementation of the collaboration environment within cross-bordered company. It allowed to setup and deploy the solution fast and economically.

References

- [1] Akao Y. 1990. Quality Function Deployment: Integrating Customer Requirements into Product Design. Portland: Productivity Press Inc.
- [2] Clausing D. 1994. Total Quality Development: A Step-By-Step Guide to World-Class Concurrent Engineering. New York: ASME Press
- [3] Dawson, D. & Askin, R.G. 1999. Optimal new product design using quality function deployment with empirical value functions. *Quality and Reliability Engineering International* 15: 17-32
- [4] Frontline Systems, Inc. Incline Village. USA. (accessed 2003), <http://www.frontsys.com>

- [5] Fung, R.Y.K, Ren, S. & Xie, J. 1996. The prioritization of attributes for customer requirement management. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Beijing, 1996: 953-958
- [6] Fung, R.Y.K, Tang, J., Tu, T. & Wang, D. 2002. Product design resources optimization using a non-linear fuzzy quality function deployment model. *International Journal of Production Research* 40: 585-599
- [7] IEEE 830:1998. Guideline for Software Requirements
- [8] Lowe, A.J. (accessed 2002). Quality Function Deployment, <http://www.shef.ac.uk/~ibberson/QFD-IntroIII.html>
- [9] Masud, A.S.M. & Dean, E.B. 1993. Using fuzzy sets in quality function deployment. *Proceedings of the 2nd International Engineering Research Conference*, 1993: 270-274
- [10] Moskowitz, H. & Kim, K.J. 1997. QFD optimizer: a novice friendly function deployment decision support system for optimizing product designs. *Computers and Industrial Engineering* 33: 641-655
- [11] Nowak, T. 2004. Synchronous collaboration systems in casting technology design. PhD Thesis. AGH University of Science and Technology. Krakow
- [12] Park, T. & Kim, K.J. 1998. Determination of an optimal set of design requirements using house of quality. *Journal of Operation Management* 16: 569-581
- [13] Pawar, K.S. & Sharifi, S. 2000. Virtual collocation of design teams: coordination for speed. *International Journal of Agile Management Systems* 2(2): 21-35
- [14] ReVelle, J.B, Moran, J.W. & Cox, C. 1998. The QFD Handbook. New York: John Wiley & Sons
- [15] Zrymiak, D. Software Quality Function Deployment. <http://software.isixsigma.com/library/content/c030709a.asp> (accessed 2004)

Change Management with Dynamic Object Roles and Overloading Views

Radosław ADAMUS^a, Edgar GŁOWACKI^c,
Tomasz SERAFIŃSKI^a and Kazimierz SUBIETA^{a, b, c}

^a *Computer Engineering Department, Technical University of Lodz, Lodz, Poland*

^b *Institute of Computer Science PAS, Warsaw, Poland*

^c *Polish-Japanese Institute of Information Technology, Warsaw, Poland*

e-mails: {radamus, tomaszek}@kis.p.lodz.pl, {edgar.glowacki, subieta}@pjwstk.edu.pl

Abstract. Database applications are an area of very specific requirements with respect to change introduction. New requirements on a database may require database schema modification or new ontology assumed for existing database entities. Such new requirements may involve necessity of changes in thousands of places of all the applications that dependent on the previous version of the database schema and ontology. In the paper we propose to relax the difficulties with the change of a database schema and ontology through two kinds of generic features that can be introduced to the object-oriented database model and to OODBMS: (1) dynamic object roles that simplify schema evolution, and (2) updateable overloading database views that introduce additional semantics to already existing semantics of database entities. The paper follows the Aspect-Oriented Programming idea, but with respect to situations that appear in database applications.

Introduction

Changes are so inseparably associated with software development that “embrace change” has become a motto for Agile Software Development methods [9]. In the change management issues we can distinguish three kinds of activities and methods:

- Activities that happen after software deployment in response to a change request. This is a traditional domain of software engineering, supported by various methodologies, standards and formalized procedures.
- Activities and methods that should be applied during software development. This concerns the quality of documentation, configuration management, modularization through well-defined interfaces between modules, layered software architectures, encapsulation and reuse principles and methods, keeping tracks between documentation and software modules, and so on. Many of these activities and methods imply additional cost of software manufacturing, thus can be questioned by software sponsors and neglected (as a rule) in situation of deadline panic.
- Features of software development environment and query/programming languages that support change management. We can mention here object-oriented programming languages and databases that assure seamless transition between software design documents (e.g. UML diagrams) and software modules (e.g.

classes) and v/v. In comparison to the previous group of methods, such features do not imply additional cost of software development and in many cases imply reduced costs.

This paper is devoted to the last kind of software change support methods. The general idea is to make already released software more flexible to possible changes. While the object-orientedness in software design and construction has made significant steps in abstraction, classification, decomposition, encapsulation and code reuse (all these features support software change), some database model and/or programming language constructs still may lead to difficulties. The biggest difficulties in software change are caused by situations when a single conceptual feature is dispersed and hidden in thousands of places across the application program code (c.f. the Y2K problem). Such cases are just the motivation for Aspect-Oriented Programming (AOP) [5] (and similar paradigms) aiming at focusing “tangled aspects” (i.e. software aspects that tend to be dispersed in many small code pieces) into continuous code modules. There are many examples of tangled aspects that are inconvenient for further software changes; among them, the AOP literature discusses security, synchronization, remoting, transaction processing, persistence and others.

The essence of tangled aspects is that some high-level software requirements or features (called “aspects”) do not fill well to a particular software production model, in particular, a database model and a database query/programming language. By changing a production model some aspects become no more tangled. For instance, aspects that are tangled in Java are no more tangled in AspectJ, a Java-like language having AOP capabilities. Current AOP ideas, however, are insufficient with respect to some requirements and features related to the database design and application programs acting on databases.

For better identification of the problem, we present simple examples. We consider the (apparently innocent) notion of *collection*, as known e.g. from the ODMG standard [11], and show that it can be the reason of problems during software change [14]. Let a *PersonClass* has a specialization *StudentClass*. Assume two extents (collections of objects) of these classes, *Persons* and *Students*. Due to the substitutability each *Student* is a *Person* too, hence we are coming to the following alternatives: (1) the collections have a non-empty intersection; (2) in all places where *Persons* are queried, a query must additionally refer to *Students*. The first alternative is not allowed in all known DBMS-s. The second alternative may require repetition the same code within many places of applications, which is just the difficulty for software change. Then assume that requirements to *Students* have been changed: each person can be a student in two or more universities. Such a change request implies dramatic changes all over the application: the inheritance must be changed into an association and all places in the code referring to *Persons* or *Students* must be adjusted. Similar dramatic changes appear after a change request requiring introducing a new *EmployeeClass* (inheriting from *PersonClass*), a new extent *Employees* and assuming multiple inheritances (i.e. a *StudentEmployeeClass* inheriting both from *StudentClass* and *EmployeeClass*). The code must involve the *StudentsEmployees* collection and must be adjusted in all places where collections *Persons*, *Students* and *Employees* are referred too. Obviously, the case is disadvantageous from the software change perspective. In the paper we discuss how the case can be smoothly handled by the notion of dynamic object roles and dynamic inheritance.

As an example of a tangled database aspect consider a database whose *Employee* objects contain the *salary* attribute. Assume that some 5 years after launching the application the requirement to this attribute has been changed: any user that reads this attribute or makes any other operation must be recorded at a special log file. In this case nothing in the database structure is changed; the change concerns only the *ontology* assumed for this attribute. The *salary* attribute is used in hundreds of places along the application code. We can suppose that references to *salary* can be hidden within dynamic SQL statements, i.e. they are not explicitly seen from the application program code. This could make the task of discovering all the places where the attribute *salary* is used extremely difficult. Note that the trigger technique in this case is inapplicable because triggers cannot be fired on read events. In classical databases the only way to fulfill this requirement is the adjustment of the code in all those hundreds of places, which can be a very long and very costly process.

We propose to cope with such cases by means of virtual updateable database views. Our concept of updateable views is similar to the *instead of trigger* views of Oracle and SQL Server, but much more general [6], [7], [8], [16]. The idea is that each generic operation acting on virtual objects (including read operations) can be overloaded by a procedure which implements the mapping of the operation to operations on stored database objects. Regarding the above example, we propose to use the view named *salary* that overloads the original *salary* attribute. Within the view one can put any additional code that does the required action. Because the name of the view is the same as the name of the attribute all the bindings to *salary* come to the view. Only the view code contains bindings to the original *salary* attribute. All the updating semantics of the original *salary* attribute can be retained or modified according to new requirements. The method allows one to make the required code change in a single place instead of the mentioned hundreds of places.

In the paper we assume the conceptual frame known as the Stack-Based Approach (SBA) [12], [13], [16]. In SBA a query language is considered a special kind of a programming language. Thus, the semantics of queries is based on mechanisms well known from programming languages, like the environment stack. SBA extends this concept for the case of query operators, such as selection, projection/navigation, join, quantifiers and others. Using SBA one is able to determine precisely the operational semantics (abstract implementation) of query languages, including relationships with object-oriented concepts, embedding queries into imperative constructs, and embedding queries into programming abstractions: procedures, functional procedures, views, methods, modules, etc. SBA and its query/programming language SBQL is implemented in many prototypes, in particular, for the XML DOM model, for OODBMS Objectivity DB, for European project ICONS, for the YAOD prototype [11] of OODBMS (where the ideas presented in this paper are experimentally implemented), and recently for ODRA, a prototype OODBMS devoted to Web and grid applications. The space limit does not allow us to present SBA and SBQL in more detail; the book [16] presents almost complete description.

The rest of the paper is organized as follows. In Section 1 we discuss the extension to the traditional object model with dynamic object roles and show that such a feature has a positive impact on the software change potential. Section 2 presents an example illustrating the change management with overloading views and presents technical details of application of overloading views for change management and query processing. Section 3 concludes.

1. The Object Model with Dynamic Roles

The object model with dynamic object roles [3], [4], [15], [16] opens new possibilities and overcomes many problems related to classical object-oriented concepts and technologies. The idea is based upon the assumption that an object consists of encapsulated, hierarchically organized roles that can be dynamically attached and detached from it. The roles make it possible to insert additional functionality into the object at runtime. A role is encapsulated, thus exists independently of other roles of the same object. Thanks to this property some classical problems of object-orientedness do not occur, e.g. name conflicts in case of multiple inheritance. Figure 1 shows an example object with roles. The object has the main role *Person* and four roles: one *Student*, two *Employee*s and one *Customer*. Even if the roles *Student* and *Employee* have attributes with the same name (e.g. incomes) no name conflict occurs because of encapsulation. Figure 1 presents the situation when one person works in two places. Note that in classical object-oriented approaches such a situation cannot be modeled through the standard inheritance; instead the database designer is forced to use associations, aggregations or some form of delegation.

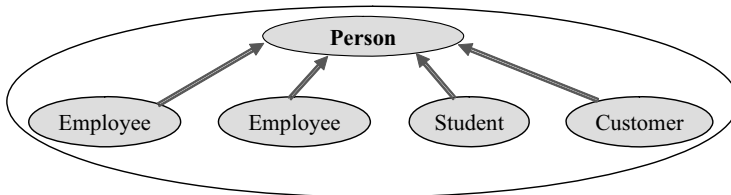


Figure 1. An object with roles

In our approach programmer can use the facility of object-oriented database model with dynamic roles in two basic protocols of the relationship between a role and its parent object:

- Forward dynamic inheritance – a parent object dynamically inherits properties of its role(s).
- Backward dynamic inheritance – a role dynamically inherits properties of its parent object [3], [4].

1.1. Motivating Example for Dynamic Object Roles

Assume example model from Figure 2. Class *PersonC* defines the object with roles (i.e. its main role). *Person* object can possess arbitrary number of *Employee* roles (i.e. she/he can work on many positions simultaneously). Due to dynamic nature of roles person can gain new, change or lose the job. This situation cannot be covered by the standard inheritance even with multiple inheritances. Additionally in contrast to multiple inheritances there are no name conflicts because each role is encapsulated entity. A *Person* object can also possess other roles, not shown here, like *Student*, *CarOwner*, *ClubMember*, etc.

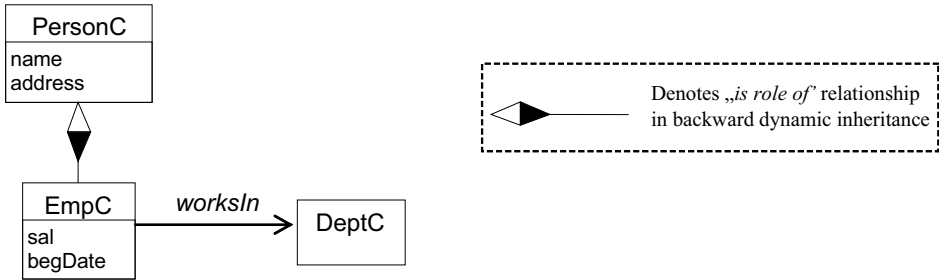


Figure 2. Example: database schema with dynamic object roles

1.2. Introducing a New Requirement

Assume that we have two different applications based on the schema presented in Figure 2. The first one is a system supporting human resources (HR) department of an organization. The second one is a computer program which facilitates submitting employee salary scheme contribution. Assume that a few years after the considered database applications have been deployed new requirements have emerged which apply the HR support system. However the modifications made to the first system should not affect the operation of the second one.

The new requirement assumes that for making summary reports and tracking profession progress the system should record not only present but also historical information about employment. To fulfill these new requirements we need to store information about previous jobs as well as any changes to the employment conditions (e.g. salary rise). Object model with dynamic roles can easily express this new property. The scenario of the modification can be as follow (see Figure 5 in the next section for example store state):

- To track the time the person works on given position we change the schema for the *EmpC* class. We add new attributes. We introduce new roles named *PrevEmp* and *PrevSal* (see Figure 3). *PrevEmp* has the same attributes as *Emp* role with addition of *endDate* attribute to store the date of changing the job. *PrevSal* has the attributes for storing previous value of the salary and the date of change.
- When the value of the salary is being updated we create new *PrevSal* role store the previous employment conditions and current as the date of change. We insert the *PrevSal* role into the *Emp* role in forward dynamic fashion (i.e. dynamic attribute). Finally we perform the update of the value of the target salary object to a new value.
- When the *Emp* role is being deleted we create new *PrevEmp* role and copy all the information form the *Emp* role (including moving all *PrevSal* sub-roles owned by the *Emp* role). We insert the *PrevEmp* role into *Person* object in backward inheritance fashion (i.e. dynamic inheritance). Finally we delete *Emp* object.

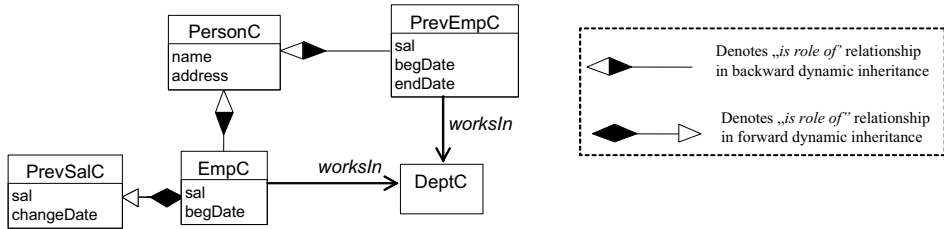


Figure 3. Changed database schema

Figure 3 presents the changed database schema. Note that the new schema implies no changes to old applications, because old database structures are not changing. In current relational, object-relational and object-oriented DBMS such a schema change will have a dramatic impact to majority of applications. This case can be treated as an example of a general rule. Dynamic object roles relax the problem with collections and substitutability that we have mentioned in the introduction. In other papers (e.g. [15]) we show many positive impacts of the dynamic role concept on the consistency, maintainability and changeability of database application software.

2. Overloading Updateable Views as a Change Management Facility

The idea of updateable views [6], [7], [8], [16] relies in augmenting the definition of a view with the information on users' intents with respect to updating operations. The first part of the definition of a view is the function, which maps stored objects into virtual objects (similarly to SQL), while the second part contains redefinitions of generic operations on virtual objects. The definition of a view usually contains definitions of subviews, which are defined on the same principle. Because a view definition is a regular complex object, it may also contain other elements, such as procedures, functions, state objects, etc. State objects make it possible to create stateful mappings, which are necessary for some purposes, e.g. security or distributed transaction processing.

The first part of the definition of a view has the form of a functional procedure. It returns entities called seeds that unambiguously identify virtual objects (usually seeds are OIDs of stored objects). Seeds are then (implicitly) passed as parameters of procedures that overload operations on virtual objects. These operations are determined in the second part of the definition of the view. We distinguished four generic operations that can be performed on virtual objects:

- *delete* removes the given virtual object,
- *retrieve* (dereference) returns the value of the given virtual object,
- *insert* puts an object being a parameter inside the given virtual object,
- *update* modifies the value of the given virtual object according to a parameter — the new value.

Definitions of these overloading operations are procedures that are performed on stored objects. In this way the view definer can take full control on all operations that should happen on stored objects in response to update of the corresponding virtual

object. If some overloading procedure is not defined, the corresponding operation on virtual objects is forbidden. The procedures have fixed names, respectively *on_delete*, *on_retrieve*, *on_insert*, and *on_update*. All procedures, including the function supplying seeds of virtual objects, are defined in SBQL and may be arbitrarily complex.

2.1. Motivating Example for Overloading Views

Because of the new security constraints all operations that are performed on salary attribute of the *Emp* role (reading and updating) has to be logged in the special log file. We can assume that the process of logging can be implemented separately (or is already present in the application).

The main problem we need to face while introducing the change is connected with its evolutionary nature. The database is used for a long time. The *Emp* role and its *salary* attribute is used in hundreds of places along the application code, possibly hidden within dynamic query statements, i.e. they are not explicitly seen from the application program code. This could make the task of discovering all places where the attribute *salary* is used extremely difficult. The trigger technique in this case is inapplicable because triggers cannot be fired on read events (which we also need to monitor because of the new security requirement). In classical databases the only way to fulfill this requirement is the adjustment of the code in all those hundreds of places, which can be a very long and very costly process.

We propose to cope with such cases by means of virtual updateable database views. Regarding the above example, we propose to use the view named *Emp* that overloads the original *Emp* role. Within the view one can put any additional code that does the required action on entire *Emp* role as well as on its attributes. Because the name of the view is the same as the name of the role all the bindings to *Emp* come to the view. Only the view code contains bindings to the original *Emp* role. All the updating semantics of the original *Emp* role and its attributes can be retained or modified according to new requirements. The method allows one to make the required code change in a single place instead of the mentioned hundreds of places.

Our method allows one to overload bindings to any object or attribute by bindings to a view with the same name. In this way views make it possible to add new semantic to all the operations (retrieve, insert, update, delete) that can be performed on the objects. Overloading views are named encapsulated database entities that can be dynamically inserted, modified or deleted. Because virtual objects delivered by an overloading view are not distinguishable from stored objects, overloading views may form a chain of views, where each next view adds new semantics to the semantics introduced by the object implementation and previous views. In this way any new requirement to a particular population of data objects can be implemented independently from other requirements. In our example it means that we can implement each of our two new requirements (and any further) as a separate view.

2.2. The Idea of Overloading Views

Views that add new semantics to (virtual or stored) database objects will be referred to as overloading views. Note that the meaning of this term is changed in comparison to the typical object-oriented terminology, which assumes that an overloading operation *m* fully substitutes the original operation *m*. In our case an overloading view adds some

specific semantics to already implemented semantics. The assumptions for the overloading views are the following:

- Stored or virtual objects named *n* in the database can be overloaded by a (next) updateable view that delivers virtual objects named *n*.
- Overloading means that after the view has been inserted all bindings of name *n* invoke the view rather than return references to objects named *n*.
- Access to the original objects named *n* is possible only inside the overloading view through special syntax.
- As in [6], [7], [8], a view has a managerial name independent from the name of virtual objects delivered by the view. The managerial name allows the administrator to make managerial operations on the views, e.g. delete a view, update it, or change its position in a chain.
- Virtual objects delivered by an overloading view can be overloaded by a next overloaded view, with the same rules. There is no limitation on the size of overloading chains.

Figure 4 illustrates this rule. Overloading views can overload simple objects (i.e. attributes of roles). In this case the natural place for an overloading view definition chain is a class. Inserting an overloading view into the given class influences the behavior of all its instances [1], [2]. Overloading can also concerns complex root objects (i.e. roles). In this case overloading chains are stored in the special root object called *DatabaseClass* (Figure 5). *DatabaseClass* introduce a level of separation that detaches overloading chains from the regular objects in the object store. In this context *DatabaseClass* plays similar role for entire object store, as the class for its instances.

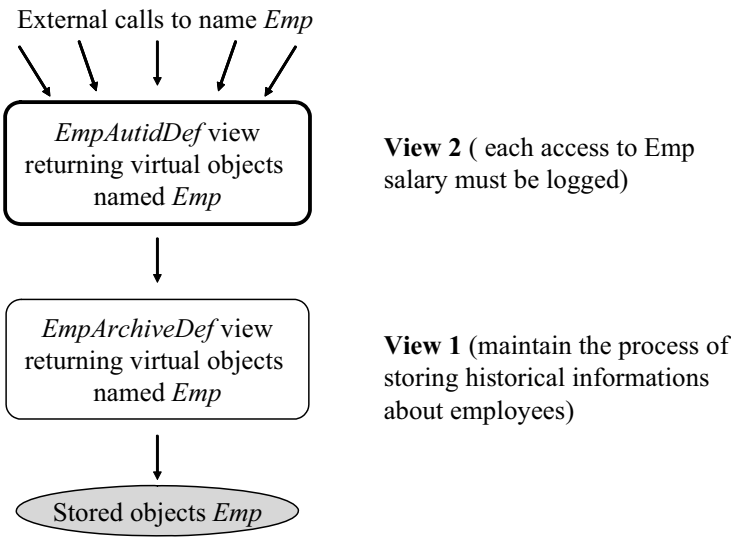


Figure 4. Example of a chain of overloading views

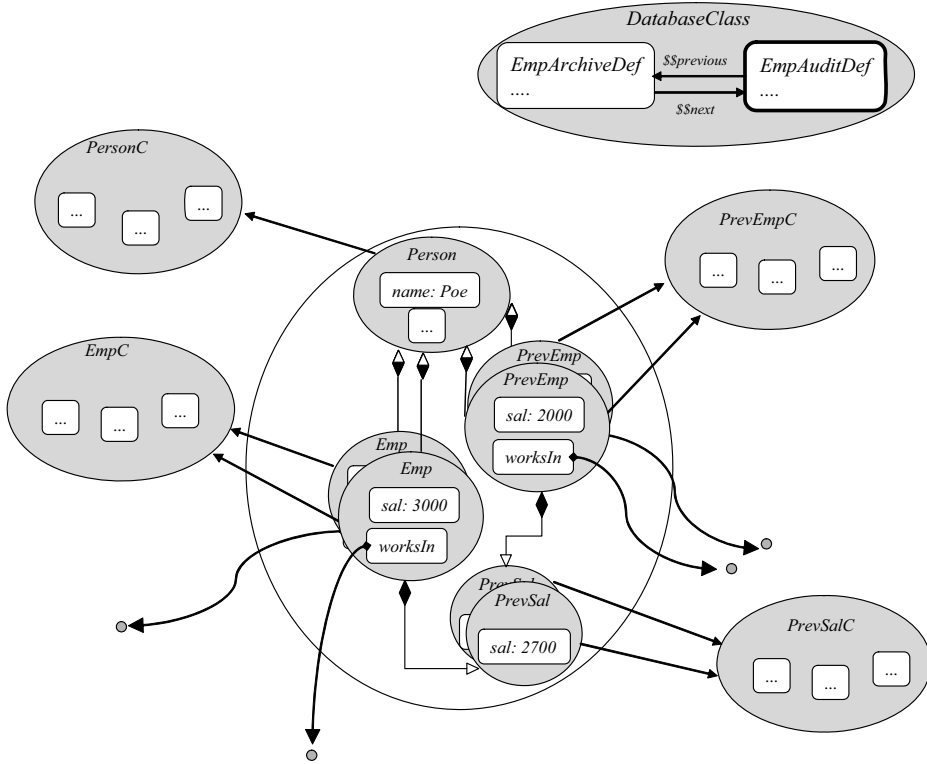


Figure 5. Example store state with chain of overloading views in *DatabaseClass*

2.3. Organization of an Overloading View Chain

A chain of overloading views has to be formed into a database structure with the following properties:

- It should be possible to find the most outer view to which all external calls are to be bound. (In Figures 4 and 5 this view is distinguished by a thicker line.)
- It should enable localizing the next view in the chain (the calling order).
- For full updating power it is also necessary to find a next view in the chain.

To implement these properties we use pointers inside view definitions. These pointers have distinguished predefined names. We assume here *\$\$previous* and *\$\$next*, where *\$\$previous* points to previously defined view and *\$\$next* v/v. The user has no possibility to use these names in programs or administrative utilities. They are only internally used by the binding mechanism. The most outer view in a chain is marked by a special flag (and has no *\$\$next* pointer).

We assume that the first element of the chain is the most outer view (*EmpAuditDef* in Figures 4 and 5). Note (Figure 5) that there is no direct connection between the chain of views and the *Emp* role. Such an explicit connection makes little sense, as it requires inserting to the last view a lot of pointers to original *Emp* roles. Thus this connection will be determined implicitly by the environment stack.

2.4. Overloading View Definition – Bindings to Original Objects

The view programmer needs the language construct that allows him/her calling the original (overloaded) object from the view body. On the syntax level of the query language the construct introduces a keyword **original**. This syntax informs the binding mechanism that the binding must be special. Below example shows the definitions of a overloading views implementing the process of storing historical information about employment and earnings of persons (*EmpArchiveDef*) and the process of logging salary access (*EmpAuditDef*) and administrative operations creating the chain of overloading views on *Emp*.

```
create overloading view EmpArchiveDef {
  virtual_objects Emp { return (original Emp) as e; }
  on_delete { insert backward role((Person)e,
    (create local (deref(e.sal) as sal, deref(e.begDate) as begDate,
      currentDate() as endDate, deref(e.worksIn) as
        worksIn)
      as PrevEmp as member of PrevEmpC );
    insert forward role (PrevEmp, e.PrevSal); //move PrevSal roles
    insert (permanent, PrevEmp); // move PrevEmp to permanent env.
    delete e; //original semantic }
  on_retrieve { return deref (e); //original semantic }
  create view SalaryDef {
    virtual_objects salary { return e.salary as s; }
    on_update(newSalary) {insert forward role(e, (create permanent
      (deref(e.sal) as sal, currentDate() as changeDate) as
        PrevSal as member of PrevSalC)); s := newSalary; }
    on_retrieve { return deref(s); //original semantic } } }

create overloading view EmpAuditDef {
  virtual_objects Emp { return (original Emp) as e; }
  on_delete { delete e; } //original semantic
  on_retrieve { return deref (e); //original semantic }
  create view SalaryDef {
    virtual_objects salary { return e.salary as s; }
    on_update(newSalary) { writeUpdateLog(); s := newSalary; }
    on_retrieve { writeDereferenceLog(); return deref(s); } } }
//administrative operations
insert EmpArchiveDef into DatabaseClass on top of chain Emp;
insert EmpAuditDef into DatabaseClass on top of chain Emp;
```

In **virtual objects** procedures in the views definition the name *Emp* is preceded by the keyword **original**. This requires binding to the previous view in the chain (according to the *\$\$previous* pointer) or to the original *Emp* object (if there is no more views in the chain).

2.5. Query Processing for Overloading Views

Overloading views require introducing specific changes to the standard binding mechanism of the stack-based approach. The changes concern the keyword **original** and bindings to stored objects overloaded by a view. The mechanism should ensure that every call to name *n* preceded with the keyword **original** causes execution of

a **virtual objects** procedure located in the view that is accessible from the given view by the $$$previous$ pointer. If the pointer is absent within the given view, the mechanism should bind the name n to stored objects n . In our example original objects (roles) are root objects while the chain of overloading views is located within a *DatabaseClass*. Binding mechanism should assure that using the name *Emp* outside the view definition will invoke the most outer view in the chain of views. To fulfill this requirement we modified the process of creating base stack sections:

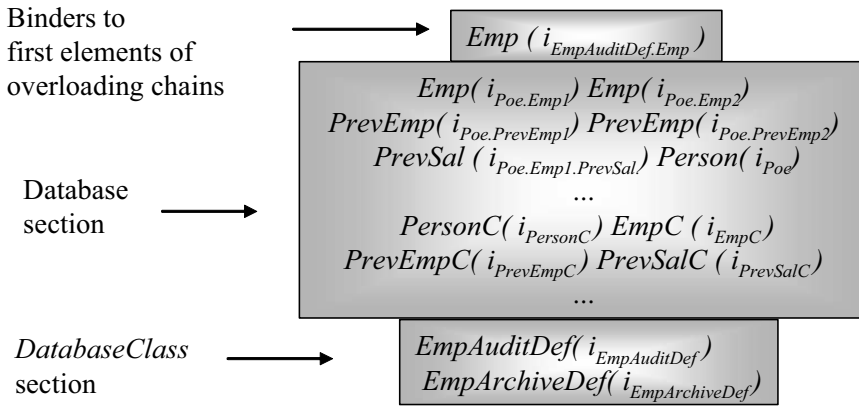


Figure 6. Modified ES for processing of chains of overloading views

Base ES state is extended with two sections: *DatabaseClass* section with binders to overloading views (for administrative operations) and the top section with the binders to the virtual objects procedures of the most outer views (see Figure 6) in the chain of views. The mechanism lets to bind properly the name *Emp*. Because the result of the binding is procedure, it is automatically invoked. Keyword original used in the virtual objects procedure definition results in calling virtual objects procedure for the previous view in the chain or getting the identifier for original object (if there is no more views in the chain) [1], [2].

3. Conclusion

We have proposed two kinds of generic database facilities that can much simplify change management of database applications. These are dynamic object roles and overloading views. Both facilities allow the programmer to introduce significant changes to the database schema and ontology without influencing existing code of applications working on the database. Additionally, the presented object model with dynamic object roles introduces expressive power for modeling historical or temporal data.

The prototype of the object database management system (ODBMS) with presented properties was implemented on the top of the YAOD database prototype [10]. Currently we advance it for the ODBMS ODRA devoted to Web and grid applications.

References

- [1] R. Adamus, K. Subieta. *Security Management Through Overloading Views*. Proc. of ODBASE, 25 - 29 October 2004, Larnaca, Cyprus, Springer LNCS 3219.
- [2] R. Adamus, K. Subieta *Tier Aspect Model Based on Updatable Views*. Proc. of the SOFSEM 2005, Springer LNCS 3381
- [3] A. Jodłowski: *Dynamic Object Roles in Conceptual Modeling and Databases*, Ph.D Thesis, Institute of Computer Science, Polish Academy of Sciences, 2003
- [4] A. Jodłowski, P. Habela, J. Płodzień, K. Subieta.: *Extending OO Metamodels towards Dynamic Object Roles*, Proc. of ODBASE, Catania, Sicily, Italy, November, 2003, Springer LNCS 2888
- [5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, J. Irwin. *Aspect-Oriented Programming*. Proc. ECOOP Conf., Springer LNCS 1241, 220-242, 1997
- [6] H. Kozankiewicz, J. Leszczyłowski, J. Płodzień, K. Subieta. *Updateable Object Views*. Institute of Computer Science Polish Ac. Sci. Report 950, October 2002
- [7] H. Kozankiewicz, J. Leszczyłowski, K. Subieta. *Updateable Views for an XML Query Language*. Proc. 15th CAiSE Conf., 2003
- [8] H. Kozankiewicz, J. Leszczyłowski, K. Subieta. *Implementing Mediators through Virtual Updateable Views*. Proc. 5th EFIS Workshop, Coventry, UK, 2003
- [9] C. Larman, *Agile and Iterative Development: A Manager's Guide*, Addison Wesley, 2003
- [10] M. Lentner, J. Trzetrzelewski: *Object-Oriented Database as an XML Repository*. Master Thesis, Polish-Japanese Institute of Information Technology, Warsaw, 2003
- [11] Object Data Management Group: *The Object Database Standard ODMG*, Release 3.0. R.G.G. Cattel, D.K. Barry, Ed., Morgan Kaufmann, 2000
- [12] K. Subieta, C. Beeri, F. Matthes, J.W. Schmidt. *A Stack-Based Approach to Query Languages*. Proc. East-West Database Workshop, 1994, Springer Workshops in Computing, 1995
- [13] K. Subieta, Y. Kambayashi, J. Leszczyłowski. *Procedures in Object-Oriented Query Languages*. Proc. 21-st VLDB Conf., Zurich, 1995, pp.182-193
- [14] K. Subieta. *Object-Oriented Standards. Can ODMG OQL Be Extended to a Programming Language? Cooperative Databases and Applications*, World Scientific 1997, pp. 459-468
- [15] K. Subieta, A. Jodłowski, P. Habela, J. Płodzień. *Conceptual Modeling of Business Applications with Dynamic Object Roles*, a chapter in the book: "Technologies Supporting Business Solutions" (ed. R. Corchuelo, A. Ruiz-Cortés, R. Wrembel), the ACTP Series, Nova Science Books and Journals, New York, USA, 2003
- [16] K. Subieta. *Theory and Construction of Object-Oriented Query Languages*. Editors of the Polish-Japanese Institute of Information Technology, 2004, 522 page

6. Knowledge Base System and Prototyping

This page intentionally left blank

The Process of Integrating Ontologies for Knowledge Base Systems¹

Jarosław KOŹLAK, Anna ZYGMUNT, Adam ŁUSZPAJ and Kamil SZYMAŃSKI
*AGH University of Science and Technology, Department of Computer Science,
Al. Mickiewicza 30, 30-059 Kraków, Poland*
*e-mails: {kozlak, azygmunt}@agh.edu.pl, alp@student.uci.agh.edu.pl,
camel_sz@go2.pl*

Abstract. Ontologies are currently more and more frequently used to represent knowledge in distributed heterogeneous environments. This approach supports knowledge sharing and knowledge reuse. In order to increase the effectiveness of such solutions, a method should be developed which would enable us to integrate ontologies coming from various sources. The article presents a concept for integration of knowledge, based on structural and lexical similarity measures, including the Similarity Flooding algorithm. The proposed concepts are demonstrated on the basis of a selected area of medical studies: the analysis of the incidence of hospital infections. Sample ontologies (exhibiting structural or lexical similarities) have been developed and for each case a suitable algorithm is proposed.

Introduction

The interest in ontologies has grown significantly with the advent of the Semantic Web. Ontologies are more and more frequently used to represent data in the distributed, heterogeneous environment offered by the WWW. Several languages have been proposed to describe ontologies. The most popular of these being currently the Resource Description Framework (RDF) [6,10] and the Web Ontology Language (OWL) [7]. Ontologies can be prepared through the Protege 2000 [12] knowledge base editing facility. The Jena [3] library enables programmers to reference knowledge stored in an ontological form. The applied ontologies consist of [19]: classes/concepts describing the classes/elements of the model domain; properties describing particular characteristics of a concept (slots, roles, properties) and restrictions applied to concept property values (facets, role description). A knowledge base is created on the basis of an ontology, augmented with sets of class instances.

1. Domain Review

1.1. Ontology Integration

The advantages of ontologies, resulting in increased effectiveness and suitability of the knowledge which they express, include the ease with which such knowledge can be

¹ This work is partially sponsored by State Committee for Scientific Research (KBN), grant no. 4T11C023 23.

shared by various applications, as well as the possibility to reuse existing knowledge components in new systems. To this end, methods and technologies for ontology integration have been created, enabling the reuse of concepts and relations determined by individual source ontologies. The ontologies undergoing integration can represent multiple types of domain knowledge and be expressed in various formats. Similar data is often expressed by differing ontologies, making their transparent integration difficult or even impossible. Ontology integration can thus be treated as a natural approach to solving the problem of ontological incompatibility.

Several methods [2,11] have been proposed for use in the integration process. *Mapping* relies on determining dependencies between individual elements (concepts, relations) belonging to separate ontologies. *Aligning* depends on achieving full convergence between several ontologies through the introduction of an ontology-wide mapping mechanism. *Merging* means developing a new ontology from two overlapping source ontologies (it is, however, required for both source ontologies to belong to the same subject domain). *Integrating* is the process of creating a new ontology which would draw upon all source ontologies through grouping, extending and specializing (this method does not require source methodologies to share a single subject domain).

The problems afflicting the process of reconciliation (understood here primarily as a means of establishing full interdependence between source ontologies) can be divided into two groups. The first group includes problems on the level of ontology description languages; the second group is related to the ontologies themselves. The causes of problems belonging to the first group typically involve syntax incompatibilities between individual languages defining each source ontology, semantic differences between identical expressions and varying language expression mechanisms (RDF, OWL).

The second group of problems can be a consequence of the naming conventions applied to the ontologies undergoing reconciliation, differences in concept structures, disparate representations of identical quantities and the usage of varying natural languages.

Our research focuses on reconciling ontologies on the semantic level. We do not concern ourselves with ontology description languages – we assume that all source ontologies are expressed using the OWL language and stored as XML files (though this is not a necessary condition).

Among the above mentioned methods of ontology reconciliation, we have decided to use mapping and aligning. The proposed approach consists of three key steps. Step 1 involves mapping individual ontology components and determining local alignment between each pair of elements belonging to both source ontologies. Step 2 (alignment) relies on globalizing this mapping through the use of the Similarity Flooding [8] algorithm. The final step requires the application of a suitable filter (or group of filters) to restrict the result derived in the previous steps.

1.2. Similarity Measures

For the purposes of the proposed integration mechanism, we have created a set of structures representing the source ontologies with the use of the Jena 2.1 interface [3]. These structures assemble graphs of source ontologies, read in from a specified file. Each graph consists of individual elements (classes and properties), reflecting the inheritance hierarchy for the source ontology and showing which properties belong to each particular class. Our graph structure also preserves the property specifications, as defined by

OWL. We differentiate between the following properties: simple (*SimpleProperty* class, representing the basic data types: int, boolean, float string); enumerated (*Enumerated-Property* class, representing data types through enumeration of their elements – the property in question contains a collection of string values constituting the enumeration) and object (*ObjectProperty* class, representing elements which reference class instances).

Both classes and properties inherit from the *AbstractConcept* class, which stores a name of a concept and an identifier which is unique with respect to the whole ontology. Each class (*ClassConcept*) also stores data on its properties, subclasses and parent class (if one exists). The *PropertyConcept* class, from which all of the previously mentioned types of properties inherit, includes a collection of classes which contain said properties. Thus, each ontology element can be referenced by either first browsing classes, then their properties, or first browsing properties, then the classes which contain them. The *OntologyGraph* class constitutes a linking element, which stores distinct lists of ontology classes and properties as well as references to the original Jena ontology trees.

The second important element of the system is the so-called similarity matrix, which reflects the proper alignment of both source ontologies and presents a basic operational structure for all ontology alignment algorithms. The columns of this matrix are indexed by elements of the first source ontology (O1), while the rows are indexed by elements of the second ontology (O2), according to their identifiers.

The first stage of the integration process consists of determining local similarities. Within the aforementioned similarity matrix, each cell (i,j) is filled by a numerical value from the $[0,1]$ range, representing the similarity of the respective elements from ontologies O1 (column) and O2 (row). The higher the value, the better the similarity between elements i and j .

In order to determine the similarity metric for each pair of elements we analyze various properties of these elements, such as names, types (in the case of properties), component values (in the case of classes) and positions in the respective inheritance hierarchies. If more than one similarity measure is applied to a given pair of elements, all the applied measures are merged using user-defined formulae with appropriate weights. It is also possible to use a user-defined thesaurus, dependent on the subject domain of both source ontologies. Below we present some of the applied measures as well as merging mechanisms. The measures can be divided into three groups: lexical, structural and complex [1].

Lexical measures: these measures are based on comparative analysis of strings representing ontology components (typically, these would be names of individual elements). Lexical similarity can be determined through Hamming or Levenstein algorithms as well as through analyzing substring similarities.

Structural measures: relying on lexical analysis alone may prove insufficient, e.g. when divergent concepts share similar names but have completely different semantics, precluding their alignment (for instance a class and a property, or two different types of properties). In such cases we can introduce structural measures which analyze such aspects as types of properties, class inheritance path or similarities between subsets of class properties. Each pair of concepts can be treated in a different manner. For instance, when a class and a property are considered, the derived value represents their structural similarity; the same occurs for two distinct properties. For two classes, however, structural similarity is evaluated as a weighted sum (utilizing one of the available

complex measures) of two criteria: the normalized optimal (in terms of local maxima) structural similarity of individual properties (the more similar the properties, the better the overall similarity) and the lexical similarity of parent class paths.

Complex measures: complex measures are used to summarize the results of several comparisons derived from the simple measures (described above). Complex measures include (for example) Minkovsky distance, weighted sum and weighted product.

Thesaurus: the thesaurus is a tool which can aid the process by determining whether a given pair of terms are in fact synonyms – if so, they are considered identical for the purposes of determining lexical similarity.

1.3. The Similarity Flooding Algorithm and Its Variants

The similarity measures presented above work on a local level, determining similarities between given pairs of ontology components (classes and properties) and generating the resultant similarity matrix. Basing on this mapping, it is possible to select these pairs of elements which exhibit the highest similarity; however it turns out that limiting ourselves to such local mappings does not produce the best results. Therefore, in order to perfect the results of ontology alignment a special algorithm, called *Similarity Flooding* (SF) [8] has been developed. The input to this algorithm is constituted by the aforementioned similarity matrix as well as by two graph structures (described below), representing both source methodologies. The algorithm operates by *globalizing* similarities and returning a new matrix with modified similarity values.

The concept behind the SF algorithm lies in the assumption that the similarity of each pair of elements is also dependent on the similarity between their respective neighboring elements. As a result, this “neighbor influence” *floods* the graph by affecting the similarity values of many adjacent element pairs.

The supporting graph structures required as input for the SF algorithm include the PCG (Pairwise Connectivity Graph) and the IPG (Induced Propagation Graph).

The nodes of the PCG represent pairs of elements from both source ontologies (i.e. elements of the Cartesian product $O_1 \times O_2$). Each pair of nodes is connected by vertex p if both elements immediately preceding and both elements immediately succeeding both nodes (pairs) in source graphs are also connected by vertex p .

It is necessary to determine the types of vertices in source graphs. In our case, source graph vertices are tagged using one of the two following symbols:

- $p = A$, if the elements connected by vertex p belong to an inheritance relationship (the first element is a subclass of the second element),
- $p = B$, if the elements connected by vertex p belong to a containment relationship (the first element is a class containing the second element, which is a property).

The next step involves the creation of the second graph – IPG. Starting from the PCG, we augment each pair of interconnected nodes with a vertex running in the opposite direction and then ascribe weights to all vertices in the graph. These weights will represent the influence which the similarity of the pair represented by a given node exerts on the similarity of the pair in the adjacent node.

Basing on both support graphs and the similarity matrix, the SF algorithm performs calculations with the aim of globalizing the discovered similarities. The algorithm itself is iterative in nature: each step consists of calculating new similarity values

for each pair of elements (i.e. each matrix cell), basing on the existing value, values in adjacent nodes (as dictated by the support graphs) and the weights attached to each relevant graph vertex for a given node. Thus, the new similarity value is a weighted sum of the inherent similarity of a given pair and of the influence exerted on this pair by adjacent pairs.

The described iterative procedure concludes when the differences between values computed in successive steps are smaller than an arbitrary threshold value for each cell of the similarity matrix (iterative convergence) or after a given number of steps (when no convergence can be achieved).

The calculation process described above constitutes the basic version of the SF algorithm. Other versions have also been developed, depending on modifications to the calculation process itself (the support structures remain unchanged). In our work we have decided to include a different variant of the algorithm (the so-called C variant), in which each new mapping constitutes a sum of the initial matrix, the matrix derived in step i and the matrix derived in step $i+1$ (which is itself a sum of the initial matrix and the matrix derived in step i). The selection of a given variant of the SF algorithm has little influence on end results, but often proves crucial for achieving fast convergence of the iterative process.

Alignment restriction through filters: the last phase of the described approach involves the selection of the final alignment basing on the similarity matrix. (By alignment we mean an arbitrary subset of cells in the similarity matrix; the entire matrix – being the largest possible alignment, containing all other alignments – is hence called a poly-alignment.)

We can thus determine various criteria for restricting the poly-alignment. The following list presents the most relevant restriction criteria [2]:

- *Cardinality constraints:* limiting the number of elements in one ontology which can be related to a single element in the other ontology – for instance, relations of type $[0,1] \rightarrow [0,1]$ do not permit relating any element to more than one element in the complementary ontology.
- *Type constraints:* rejecting all relations which pair elements of differing types.
- *Thresholds:* only considering these pairs of elements which exhibit sufficiently high similarity.
- *Maximal matching:* this criterion selects (from several variants) the variant which maximizes total alignment.
- *Stable marriage:* selecting the alignment which does not contain two pairs (x,y) and (x',y') such that $\text{sim}(x,y) > \text{sim}(x,y')$ and $\text{sim}(x,y') > \text{sim}(x',y')$ or $\text{sim}(x',y) > \text{sim}(x,y)$ and $\text{sim}(x',y) > \text{sim}(x',y')$.

Our algorithm relies on a combination of the above criteria, implemented as two filters. The first filter is called *best matching* and it involves successive selections of the best mapping (i.e. the largest element in the poly-alignment). Once this best element is found, the matrix is stripped of the corresponding column and row, whereupon the entire procedure is repeated for all remaining columns or rows (depending on which source ontology contains less elements). This algorithm produces an alignment which satisfies the $[0,1] \rightarrow [0,1]$ criterion. Notwithstanding the quality of individual alignments, the number of the selected pairs remains stable and equal to the number of elements in the smaller ontology. The second filter, called *threshold matching*, is based on asymmetrical dependent similarities. For each element of ontology O1 we locate the most

similar element of ontology O2 (i.e. the row in which the similarity value for a given column is largest). Subsequently, all other similarities in a given columns become *dependent similarities* and their values are divided by the located maximal value. This procedure is repeated for each column, and an analogous procedure is also conducted for all rows in a second instance of the similarity matrix (the O2 perspective). The next step is to apply an arbitrary threshold in the $[0,1]$ range, determining which elements in the poly-alignment should be replicated in the final alignment constituting the result of the filtering algorithm. The dependent similarities method preserves the stable marriage criterion, although it does not guarantee preserving the $[0,1]$ – $[0,1]$ criterion (however, the probability of satisfying this criterion grows as the threshold value approaches 1.0).

2. Applied Ontologies

In order to test the correctness of the proposed solution, we have created three ontologies representing a fragment of knowledge in the hospital infections domain. The authors are quite familiar with this field, owing to their longstanding cooperation with the faculty of the Jagiellonian University Medical College as well as the Polish Hospital Infection Association for the purposes of creating a hospital infections database and an expert system for diagnosing and treating various types of hospital infections.

The test ontologies have been engineered in such a way as to stress the differences between individual alignment methods. Hence, the first ontology (O1) is lexically different (i.e. it uses different naming) but structurally similar to the second ontology (O2) which in turn remains structurally different but lexically similar to the third ontology (O3). As a result, ontology O3 is both lexically and structurally different from ontology O1. All test ontologies have been created using the Protege 3.0 tool, then converted to the OWL language before used as input for the SF algorithm.

The proposed ontologies contain relationships which arrange objects into *possession* hierarchies. Thus, the first ontology, the *person* afflicted (“person possesses infection”) by the *infection* is treated (“person possesses therapy”) with a *therapy*. One *therapy* can involve numerous *procedures*. Each of these elements has a number of characteristic properties. Ontology O2 uses a similar structure, but different naming than ontology O1. All ontologies are thus differentiated with respect to classes and properties. The third ontology has a very different structure than the two previous ones, but it is also much simpler and thus less flexible. Its naming is similar to that of ontology O2.

3. Basis System Architecture Model

The system should provide an infrastructure enabling access to knowledge derived from knowledge sources. In order to maximize the usefulness of knowledge, we need to integrate all source ontologies – this process will enable us to utilize the largest possible repository of knowledge during reasoning and knowledge discovery.

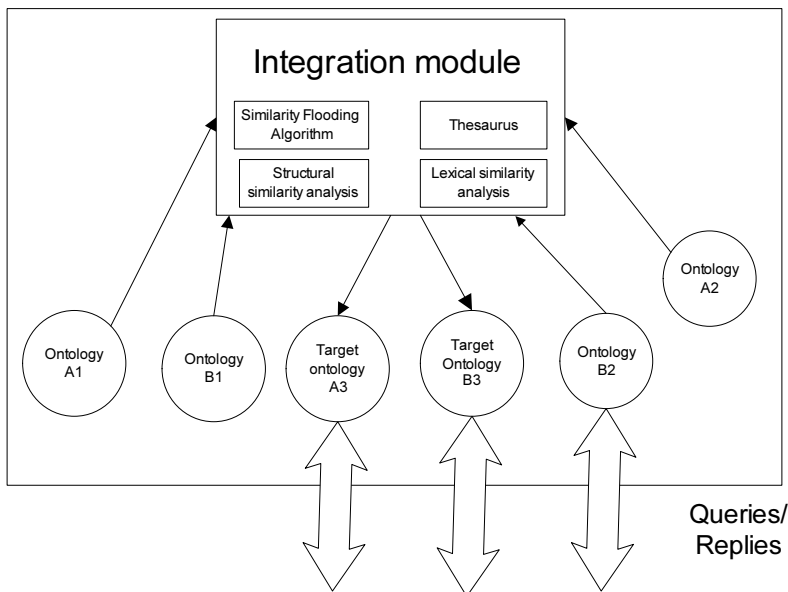


Figure 1. Distributed ontologies and the process of their integration.

The basic architecture of the implemented environment (Fig. 1) includes the following components:

- Query interface,
- Source and resultant ontologies stored in text files (OWL, RDF) and available over the Internet. Each ontology is supplemented by keywords, facilitating the selection of ontologies for integration,
- The ontology integration module, with the following submodules: structural similarity evaluation module, lexical similarity evaluation module, thesaurus, Similarity Flooding actuation module.

4. Semiautomatic Ontology Integration Algorithm

Basing on the tests conducted for selected types of ontologies we have proposed an algorithm to steer the ontology integration process. First, we determine the alignment of individual elements using simple structural and lexical measures, then – if need be – we apply the SF algorithm.

We also distinguish two stages of the algorithm preparation process: global configuration (common to all types of source ontologies) and local configuration (dependent on the choices made by the user with regard to which similarity measures should be considered when integrating ontologies).

Global configuration: Global configuration involves lexical, structural and global aspects as well as setting up the SF algorithm. This stage also includes attributing weights to component measures. Setting up the SF algorithm requires selecting the threshold value, the epsilon value (minimal iterative improvement for which another pass should be executed) and the maximum number of iterations.

Variant configuration: Variant configurations depend on the types of source ontologies and involve varying weights for lexical and structural measures as well as varying methods of applying the SF algorithm and poly-alignment restriction filters. In our work we consider relations between the following pairs of ontologies:

- Lexically similar but structurally different ontologies (L+S−). This case requires attributing significant weights to lexical measures (e.g. 0.9), while depreciating structural ones (e.g. 0.1). Owing to the lack of structural similarities, applying the structurally-oriented SF algorithm is not recommended.
- Structurally similar but lexically different ontologies (L−S+). During initial alignment higher weights have to be attached to structural measures than to lexical ones. The SF algorithm goes a long way towards improving the end result. If it is imperative to minimize the number of erroneous mappings, the basic version of the algorithm with the *threshold matching* filter works best. On the other hand, if the goal is to maximize the number of correct mappings (while also allowing for a higher percentage of incorrect ones), the basic version of the algorithm with the *best matching* filter appears to be the most suitable.

5. Results

We have tested the behavior of the algorithm for selected pairs of ontologies sharing a subject domain, but exhibiting significant differences with respect to lexical and/or structural aspects. The results of the algorithm are expressed as pairs of elements, representing the final alignment for a given pair of ontologies. In order to verify the correctness of such alignments we have prepared lists of proper mappings, then compared them with the results of our algorithm. For each final alignment of a pair of ontologies we have defined two measures characterizing the performance of the algorithm: conditional performance (i.e. the number of correct mappings divided by the total number of resultant mappings) and unconditional performance (the number of correct mappings divided by the total number of correct mappings in the reference set).

The graphs presented below depict the success rate of initial alignment measures as well as the performance of the SF algorithm with its attendant filters for a given configuration of source ontologies. We thus present:

- the average improvement in mapping results following the application of the SF algorithm (Figs 2, 4) using various filters and variants of the algorithm, with respect to the initial alignment (no SF algorithm, filters applied),
- the average final alignment (Fig. 3) – using various filters and variants of the SF algorithm.

The SF algorithm does not improve alignment for structurally different ontologies because it is by design focused on structural aspects (the similarity of each pair of elements is influenced by the similarities of adjacent pairs in the ontology structure).

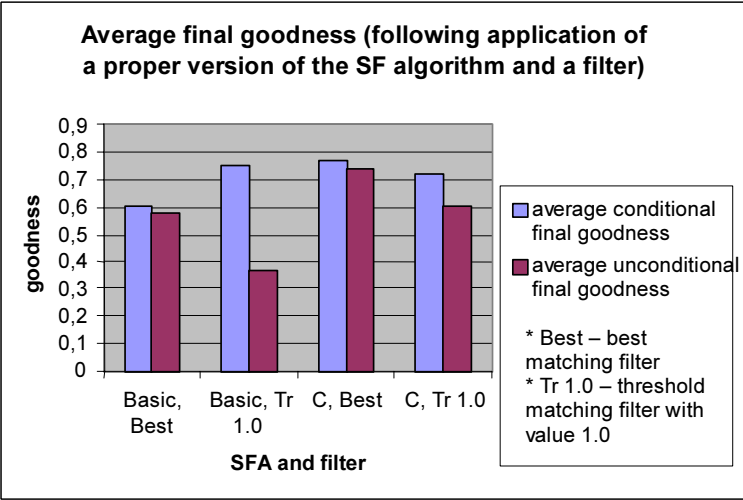


Figure 2. Structurally different, lexically similar ontologies. Average final goodness (following application of a proper version of the SF algorithm and a filter).

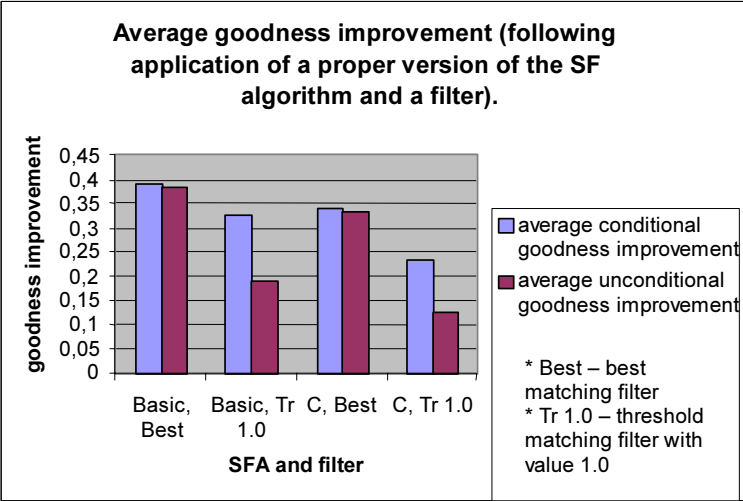


Figure 3. Structurally similar, lexically different ontologies. Average goodness improvement (following application of a proper version of the SF algorithm and a filter).

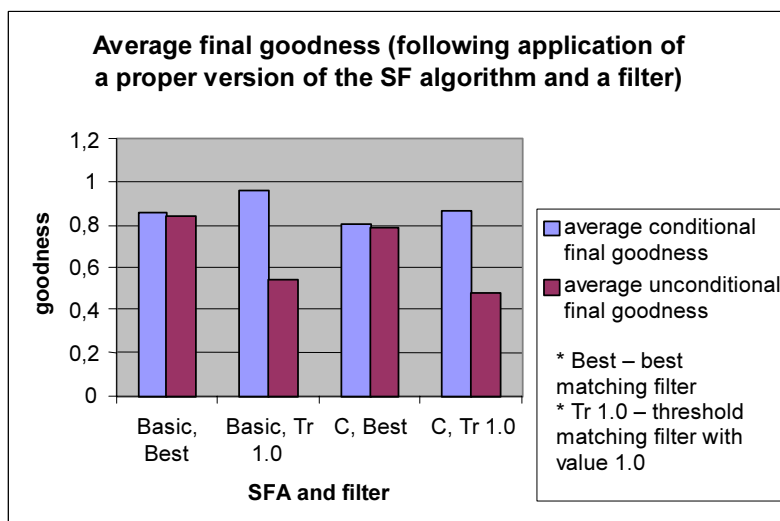


Figure 4. Structurally similar, lexically different ontologies. Average final goodness (following application of a proper version of the SF algorithm and a filter).

Comparing the results yielded by the algorithm for lexically similar but structurally different ontologies (Figs 3, 4) points to the conclusion that both variants of the SF algorithm (basic and C) are fairly similar, although the C variant seems to perform slightly better for pairs of lexically similar ontologies, while the basic version seems preferable for ontologies that are structurally aligned (significant improvement has been observed). It follows that variant C is inherently more stable and its results aren't significantly different from the initial alignment. Furthermore, the added advantage of the C variant is its faster convergence.

The comparison of filters is dependent on the selected evaluation criteria. When considering conditional and unconditional performance metrics (described above), we can show the inherent differences between both of these criteria, especially for the *threshold matching* filter. This is due to the fact that – contrary to the *best matching* filter – the *threshold matching* filter may attempt to align pairs of elements multiple times, thus raising the value of the unconditional metric.

The SF algorithm is better applied in its C variant than in its basic variant due to better stability and faster convergence. However, if the source ontologies exhibit significant structural differences, the algorithm should not be used at all. The selection of filters depends on the type of the required alignment (unary alignment for *best matching*, stable marriage criterion for *threshold matching*).

Figures 5, 6 and 7 depict the stages of integrating ontologies which are lexically different but structurally similar. Figure 5 presents the reference alignment, which the algorithm should approximate. Following the application of the *best matching* filter we arrive at the structure presented in Fig. 6, not totally consistent with the reference alignment. However, applying the SF algorithm (Fig. 7) results in further improving the quality of the solution.

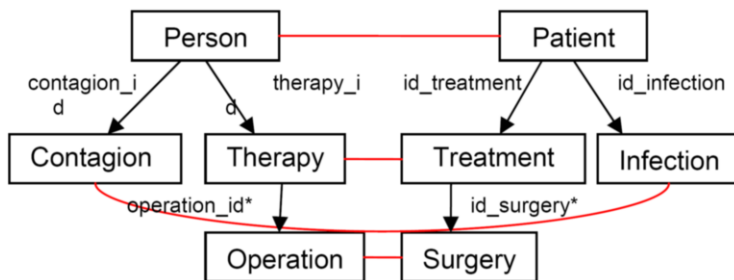


Figure 5. Reference alignment for structurally similar ontologies.

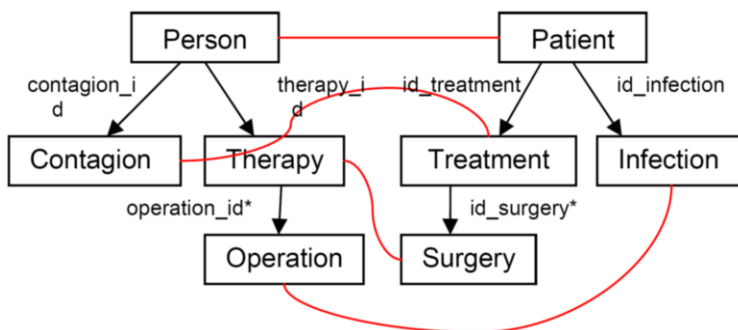


Figure 6. Automatic alignment.

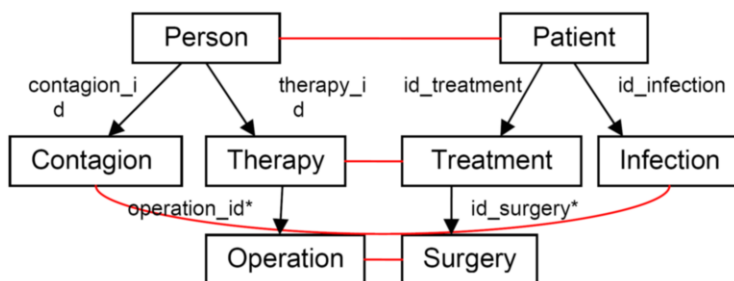


Figure 7. Alignment improved by the SF algorithm.

6. Conclusions

The research presented in this paper constitutes a stage in the development of a system whose aim will be to merge various ontologies, enabling automatic integration of knowledge as a response to emerging user needs [4,5,13]. The paper presents several methods and algorithms for semiautomatic ontology integration, minimizing human involvement in the process. Experiments have shown that it is possible to adjust the algorithm for the purposes of integrating lexically or structurally similar ontologies. Further research will focus on extending the scope of the integration algorithm to cover

other types of ontologies and to implement reasoning modules which will make use of the resulting merged ontologies.

References

- [1] Euzenat J., Le Bach T., Barrasa J., Bouquet P., De Bo J., Dieng R., Ehrig M., Hauswirth M., Jarrar M., Lara R., Maynard D., Napoli A., Stamou G., Stuckenschmidt H., Shvaiko P., Tessaris S., Van Acker S., Zaihraye I. State of the art on ontology alignment, Knowledge Web Deliverable, Technical Report, INRIA, 2004.
- [2] Fridman N., Musen, M.: SMART: Automated Support for Ontology Merging and Alignment, Twelfth Workshop on Knowledge Acquisition, Modeling, and Management, Banff, Canada, 1999.
- [3] JENA – A Semantic Web Framework for Java, <http://jena.sourceforge.net/index.html>.
- [4] Kozlak J., Zygmunt A.: Integration of results of data-mining processes using agent approach, Proceedings of Symposium on Methods of Artificial Intelligence, AI-METH-2003, Gliwice, Poland.
- [5] Kozlak J., Zygmunt A.: Zaawansowane metody reprezentacji wiedzy z wybranego obszaru medycyny, Andrzej Kwietniak, Andrzej Grzywak (eds.), Współczesne problemy sieci komputerowych. Zastosowanie i bezpieczeństwo, *WNT* 2004.
- [6] Manola F., Miller E.: RDF Primer, W3C Recommendation, <http://www.w3.org/TR/2004/REC-rdf-primer-20040210>, 10.02.2004.
- [7] McGuinness D.L., Harmelen F.: OWL. Web Ontology Language. Overview, W3C Recommendation. <http://www.w3.org/TR/2004/REC-owl-features-20040210>, 2004.
- [8] Melnik S., Garcia-Molina H., Rahm E., Similarity flooding: a versatile graph matching algorithm and its application to schema matching, *Proceedings of the 18th International Conference on Data Engineering* (ICDE '02), San Jose, Ca, 2002.
- [9] Noy N.F., McGuinness D.L.: Ontology Development 101: A Guide to Creating Your First Ontology, *SMI Tech. Report* SMI-2001-0880.
- [10] Resource Description Framework (RDF), <http://www.w3.org/RDF>.
- [11] Tamma V.: An Ontology Model supporting Multiple Ontologies for Knowledge sharing, Thesis of University of Liverpool, 2001.
- [12] Welcome to the Protege Project, <http://www.w3.org/RDF/>.
- [13] Zygmunt A., Kozlak J.: Metody efektywnego zarządzania wiedzą a reprezentowaną a w postaci ontologii, Zdzisław Szyjewski, Jerzy S. Nowak, Janusz K. Grabara (eds.) Strategie informatyzacji i zarządzanie wiedzą, *Wydawnictwo Naukowo-Techniczne*, 2004.

Designing World Closures for Knowledge-Based System Engineering

Krzysztof GOCZYŁA, Teresa GRABOWSKA,
Wojciech WALOSZEK and Michał ZAWADZKI
*Gdańsk University of Technology,
Department of Software Engineering,
ul. Gabriela Narutowicza 11/12, 80-952 Gdańsk, Poland
e-mails: {kris, tegra, wowal, michawa}@eti.pg.gda.pl*

Abstract. With development of knowledge-based components software engineers face new challenges to integrate knowledge-based components with other ones. This task requires defining a communication language between components and a process of interchanging information. The paper focuses on the conversation between knowledge-based and data-based components. A language to define Data View Manifest, describing data required by a data-based component, is specified. Moreover, the way of transforming required knowledge into data by a knowledge-based component is defined.

Introduction

In the past decade software engineers have been faced with rapid development of knowledge processing ideas and techniques. Variety of available knowledge-processing tools and components, offering different capabilities and having different limitations, connected with not always fully recognized distinction between knowledge and data are a continual source of difficulties.

Authors of this paper personally experienced that embedding knowledge processors into large software systems can be a very confusing task. This paper tries to summarize their experience gained in the course of development of PIPS project [1], [2]. PIPS (*Personal Information Platform for life and health Services*) is a 6th European Union Framework Programme project whose main goal is to create a Web infrastructure to support health and promote healthy life style among European communities.

In the course of the project we have determined that three issues were the major source of confusion among software engineers in the project, namely: the difference between knowledge and data, the role of knowledge-processing tools in the system as a whole, and (last but not least) integration of knowledge-based components with “traditional” ones.

We address all the three issues in this paper. The former two are addressed very briefly due to the fact that many papers on these topics already exist [3], [4]. However we tried to summarize the question from a practical point of view, pointing out the arguments that turned out to be most convincing for software engineers. The third issue is addressed in more details. The reason is that, to the best of our knowledge, hardly discussed is this problem in the available literature.

The question of integration of components is often critical for achieving success. Choosing, aligning, and accepting a proper way of integration can be a very hard task in a project environment with engineers having experience level in knowledge processing varying from very high to almost zero. To overcome this difficulty we have created a flexible language for expressing “needs for knowledge” of different parties involved in a project. The language is simple enough to be used by people with minimal, or even none, theoretical background, and is formal enough to be used as a base for automatic design of data structures and processes needed for proper integration. Moreover, we argue that the major part of the communication process should be rest on knowledge-based components, in this way releasing the designers of “traditional” components from necessity of gaining in-depth understanding of knowledge engineering.

The rest of the paper is organized as follows. Section 1 describes differences between Open World and Closed World Assumptions from a practical point of view. In Section 2 we give several proposals of ways of embedding knowledge-based components in a software system. In Section 3 we present a novel technique of formulating manifests describing knowledge exploited by various parties in a project and propose some techniques of making this knowledge available to them. Section 4 summarizes the paper.

1. Knowledge-Driven Software Engineering

Knowledge-based components process ontologies [3]. An *ontology* is a description of some domain of interest, which is sometimes called a *universe of discourse* or simply a *universe*. An example of such domain is: medicine, nutrition, etc. A vital component of any knowledge-based system is a knowledge base, which is a physical component responsible for storing and processing ontological data. A knowledge base can be compared to an encyclopedia which organizes its contents (an ontology) in a way which makes it available to readers.

One of main tasks of a knowledge-based system is inference. On the basis of known statements the system derives new ones, which have not been explicitly stated. The process of inference takes place in answering queries. As a consequence, also implicit statements are taken into consideration in query answering.

Knowledge-based systems use so-called Open World Assumption [5], which basically means that they do not consider gathered data complete. For example, if a knowledge base does not have any information about Mary’s children, the system does not assume that Mary is childless, but considers all possible options in the process of inference. The Open World Assumption (OWA) differentiates between a knowledge base and a database (the latter exploiting so-called Closed World Assumption (CWA)).

According to our experience, the OWA is a major source of misconception among software engineers. Consider the following scenario: A knowledge base is populated with the following information: *Mary is a woman. Jane is a woman. Kate is a woman. Mary is a mother of Jane. Mary is a mother of Kate.* Then the following query is issued: *List all people who have only daughters.* Properly implemented OWA component should **not** include Mary in the list. What is more, it should also **not** include Mary in the answer to the following query: *List all people who do not have only daughters.*

The reason behind this is as follows: the component does not know whether the information it possesses about Mary is complete, i.e. it is not certain that there is no son of Mary the component does not know about. However rational, this behaviour of the component turns out to be very confusing to software engineers due to obvious discrepancy with behaviour of relational databases.

The OWA requires the inquirer to carefully consider her question and to determine whether she asks about e.g. people for whom it is certain that they have only daughters, or people about whose sons we do not have records, or people for whom it is possible that they have only daughters. Introduction of those modalities substantially increases system capabilities. A knowledge-based system according to OWA should be able to precisely identify areas of our confidence and our ignorance, and to consider various alternatives in decision-taking processes.

The above does not imply that the OWA components totally supersedes components based on the CWA. CWA components have many advantages, one of them is the obvious gain of performance we can achieve by ignoring alternatives. Sometimes considering all possible options may be harmful to decision-making process, making it imprecise, inefficient or even inutile. In these cases we have to make some assumptions about the state of the universe, i.e. to “close our world” by designing a proper *world closure*.

We argue that careful and proper identification of areas of a system in which OWA should be used (i.e. modalities should be taken into consideration) and areas in which some kind of world closure should be used, along with design of interfaces (“bridges”) between these two parts, is an important task in knowledge-based system engineering. In the further part of the paper we provide justification of this argument along with proposals of some techniques which are focused on facilitating this task.

2. Open World vs. Closed World Conversations

The problems described in the previous sections have appeared in the course of PIPS project. PIPS concentrates on providing health-related assistance to its users: citizens, patients and healthcare professionals. PIPS serves citizens and patients by helping them in making decisions concerning their everyday practices and deciding whether they should consult their doctors. PIPS can also be used by healthcare professionals and can provide them with health-related knowledge and advise about course of therapies.

The heart of the PIPS system are two subsystems (for more architectural details refer to [1]): the Decision Support Subsystem (DSS) and the Knowledge Management Subsystem (KMS). DSS consists of a set of agents that are responsible for interaction with users. To fulfil users’ requests DSS agents may have to exploit health-related knowledge bases managed by KMS (see Figure 1).

KMS in PIPS is constructed in accordance with OWA. Knowledge-based components (depicted on the right-hand side of Figure 1 with a symbol of brain inside them) are designed in the way that they do not consider data they store complete. They are also “OWA-aware inquirers” that are able to formulate queries that precisely describe desired results (for this purpose, the special language extending DIG 1.1 [6], called DIGUT [7], has been created in the course of the PIPS project). Consequently, the knowledge-based components can seamlessly exchange information with each other without the risk of misunderstanding.

On the other hand, DSS consists of components that are in majority “traditional” (depicted on the left-hand side of Figure 1 with a symbol of a relational database inside them). Of course it is desirable (even for safety reasons) that some of DSS agents are aware of OWA. Such agents can communicate directly with KMS by formulating adequate DIGUT queries (like the component at the top left of Figure 1).

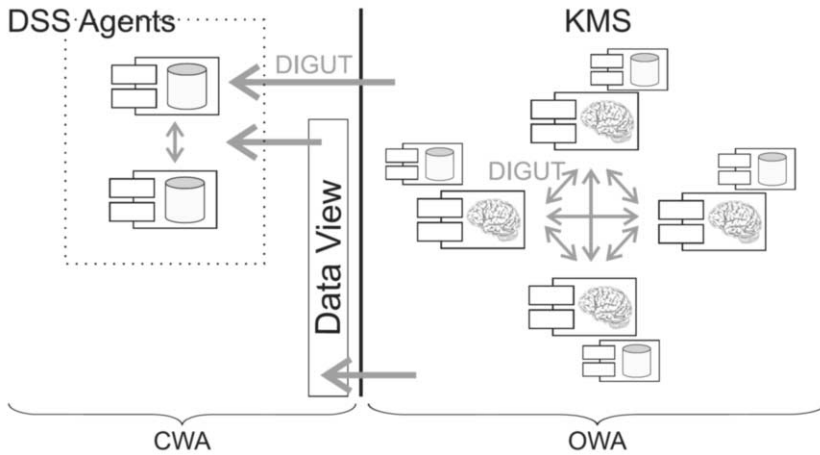


Figure 1. The PIPS system decomposition

This way of communication of DSS with KMS has been implemented in PIPS but turned out to be in many cases inconvenient. The reason is that many DSS agents need to exploit only some part of the knowledge managed by KMS. Putting on all of them the requirement of OWA-awareness is a source of excessive work for DSS developers.

The above observation has led us to forging an idea of Data Views solution. A Data View is a *world closure*, i.e. a part of knowledge freed in some way by KMS from “modalities of certainty” and encoded in the form of a relational database. In this way most of DSS components may stay unaware of OWA. Specification of way of preparation of a specific Data View is encoded in a simple language (described in Section 3) use of which does not require in-depth understanding of knowledge management-related issues. What is more, the performance of the system is expected to increase as there are some tasks that are inherently executed much more efficiently by relational databases (like calculating sums or averages).

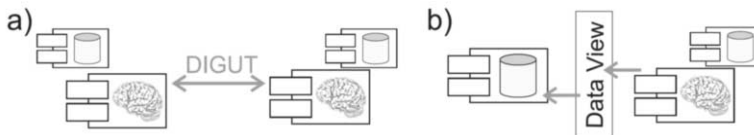


Figure 2. Mind-to-Mind and Mind-to-Data ways of communication

The proposed solution can be applied to a broad range of systems and has its precise and common sense explanation. While analyzing communication process between intelligent and non-intelligent creatures in the real world, it can be noticed that

the language of communication is quite different. The intelligent communicate with each other in one way, but the non-intelligent in a quite different manner. Moreover, the intelligent communicate with the non-intelligent in a way understandable for the non-intelligent. In such a conversation the intelligent cannot use the whole available means of expressiveness because they would not be understood.

This deliberation can be directly applied to communication between data-based and knowledge-based components. As for now the most understandable language for data-based components is SQL, so the knowledge-based components should express their knowledge transforming it to relational data. Such a Mind-to-Data approach is depicted in Figure 2b.

It can be assumed that in some systems all components that take part in a conversation are knowledge-based. Such an approach is depicted in Figure 2a as a Mind-to-Mind approach. In this approach the language of communication is very expressive and suitable for intelligent components.

Although possible, appliance of Mind-to-Mind approach in a whole system seems to be a purely theoretical solution. It is not possible, even for very intelligent creatures, like people, not to use calculators. Knowledge-based component will always take advantages of using data-based components, that can carry out some data operations more efficiently. Due to this fact, the problem of interfacing OWA and CWA is expected to gain importance in the foreseeable future.

3. Data View Approach

The Data View approach assumes that there are at least two components of a system: a knowledge-based component that processes some ontology, and a client component interested in some part of the knowledge. The client does not want to be aware of all OWA modalities. For the client the most convenient way of accessing data is use of SQL queries. The knowledge-base component is responsible for preparation of the required data (a world closure) and loading it into a database (as a Data View).

The Data View approach can be embedded into a stage analysis or design of a software project phase. The course of actions in the Data View approach is as follows. The specification of area of interest of the client component has to be prepared (either by designers of the component or a person responsible for integration of this component). This specification is called a *manifest*. On the basis of the created manifest a database schema is generated (this process can be automated or semi-automated). The two phases are described in the following subsections.

3.1. Data View Manifest

To allow users to express their domain of interest we propose a language for expressing “needs for knowledge”. The language, called NeeK [10], is based on the following assumptions:

- simplicity – the language should be simple enough to be understandable by the broadest possible range of people,
- generality – the information expressed in NeeK should be general enough to be used by various knowledge-based components.

Following the above, a generic model of an ontology has been assumed. An ontology consists of objects called *individuals*. Individuals can be related with each other with *roles*. Definitions of roles specify their *domain* (a set of objects – *role subjects* – that can appear on the left-hand side of any related pair) and *range* (a set of objects – *role fillers* – that can appear on the right-hand side of any related pair). Individuals may also have *attributes* describing their features. An attribute can be perceived as a functional role with range being a concrete domain (e.g. String, Integer). An ontology also maintains a hierarchy of object categories called *concepts*. This generic model conforms to Description Logics paradigm (DL) [5]. The exemplary DL ontology is depicted in Figure 3.

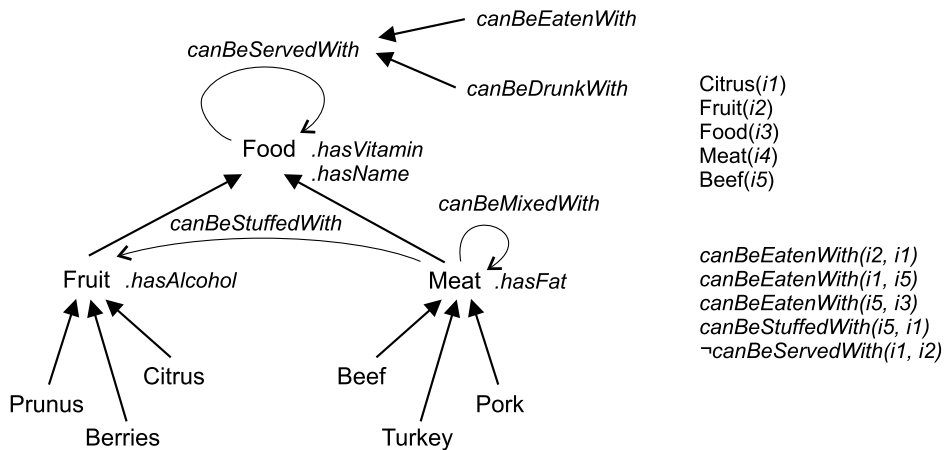


Figure 3. An exemplary ontology. Concept names are denoted with bold typeface, role names with bold-italic typeface and attribute names with italic typeface. Thin arcs represent roles, thick arrows represent concept and role inheritance

A Data View Manifest can be built from any number of statements. There are two kinds of statements: INTERESTED IN and NOT INTERESTED IN. With use of these statements we include or exclude sets of ontology entities (i.e. concepts, roles, attributes) from the domain of interest of the client. A general rule is that any statement takes precedence over previous ones.

In the following we define meaning of manifest statements. All examples included in the explanations refer to the ontology from Figure 3.

3.1.1. Interested in Concept

The statement INTERESTED IN CONCEPT is used by a client to specify its interest in the set of individuals belonging to the specified concept. Additionally, for the convenience of a creator of a manifest this statement implicitly declares that the client is also interested in: values of all attributes of these individuals and information about all instances of all roles describing how these individuals are related to each other. The statement has one required parameter *concept_expression* that defines the concept (atomic or complex) to which individuals of interest belong.

If the statement INTERESTED IN CONCEPT appears without any additional keywords it denotes that the client is interested in the set of individuals for which it is

certain that they are instances of the specified concept¹. The example of such a situation is the statement INTERESTED IN CONCEPT *Fruit* meaning that the domain of interest encompasses all individuals of *Fruit* concept, values of attributes defined for these individuals, i.e. *hasVitamin*, *hasName* and *hasAlcohol* and the relation *canBeServedWith* between these individuals. It is worth noticing that the client implicitly expresses its interest in all attributes, defined for *Fruit* concept and inherited from *Food* concept. The same rule applies to roles, thus the role *canBeServedWith* and its descendants *canBeEatenWith* and *canBeDrunkWith* belong to the domain of interest. The set of individuals the client is interested in are *i1* and *i2*. For both of these individuals the knowledge base knows (is certain) that they are fruit. Individual *i1* is a citrus, and with respect to terminology each citrus is a fruit. Individual *i2* is a fruit. About any other individual the knowledge base cannot be certain that it is a fruit.

The statement can appear with additional keywords: POTENTIALLY and HIERARCHY. Each keyword widens the domain of interest in the way described below.

The keyword POTENTIALLY widens the client interest by the set of individuals which **may possibly** be instances of the specified concept. It means that the set of individuals the client is interested in encompasses those individuals for which it is certain or possible that they are instances of the specified concept². For example, the statement INTERESTED IN CONCEPT POTENTIALLY *Fruit* means that the domain of interest encompasses all certain and potential individuals of *Fruit* concept, values of attributes defined for these individuals, i.e. *hasVitamin*, *hasName* and *hasAlcohol* and the relation *canBeServedWith* between these individuals. The individuals the client is interested in are *i1*, *i2* and *i3*. The membership of the individuals *i1* and *i2* is obvious. Individual *i3* belongs to the domain of interest because it can be a fruit. Everything what the knowledge base knows about individual *i3* is its membership to *Food* concept. This knowledge does not exclude that the *i3* can be a *Fruit*.

The keyword HIERARCHY widens the client interests by the information about membership of these individuals to subconcepts of the specified concept. For example, the statement INTERESTED IN CONCEPT HIERARCHY *Fruit* means that the client is interested not only in the fact that *i1* and *i2* are members of *Fruit* but also in the fact that *i1* is a citrus and in the fact that knowledge base does not know to which subconcepts of *Fruit* *i2* belongs.

3.1.2. Interested in Role

The statement INTERESTED IN ROLE is used by a client to specify its interest in the set of pairs of individuals that are related to each other with the specified role. The statement has one required parameter *role_expression* that defines the role (atomic or complex) the client is interested in.

If the statement INTERESTED IN ROLE appears without any additional keywords it denotes that the client is interested in all pairs of individuals for which it is **certain** that they are related to each other with the specified role. The example is the statement INTERESTED IN ROLE *canBeServedWith* meaning that the client is interested in pairs of individuals for which it is certain that they are related to each other with the role *canBeServedWith*. Such pairs are (*i2*, *i1*), (*i1*, *i5*), (*i5*, *i3*). All these

¹ DL semantics of this set of individuals can be expressed as $\mathbf{KC}([5]$, Ch. 2.2.5)

² DL semantics of this set of individuals can be expressed as $\mathbf{K-K-C}([5]$, Ch. 2.2.5)

pairs are related with the role *canBeEatenWith* that is subsumed by the role *canBeServedWith*.

Additionally, for the INTERESTED IN ROLE statement the client can specify two optional parameters that denote to which concepts related individuals must belong. Concepts of the role subjects and the role fillers are specified separately. For example, the statement INTERESTED IN ROLE *Meat canBeStuffedWith Fruit* means that the client is interested in pairs of individuals for which it is certain that they are related to each other with the role *canBeServedWith* and additionally the role subject must belong to *Meat* and role fillers must belong to *Fruit*. There is only one such pair (*i5*, *i1*). The pair (*i2*, *i1*) is excluded because *i1* is a fruit not a meat and the pair (*i5*, *i3*) is excluded because it is not certain that *i3* is a fruit.

The statement can appear with additional keywords: POTENTIALLY, HIERARCHY, WITH DOMAIN and WITH RANGE. Keywords POTENTIALLY and HIERARCHY widen the domain of interest analogically as for INTERESTED IN CONCEPT statement. Keywords WITH DOMAIN and WITH RANGE narrows the domain of interest in the way described below.

The keyword WITH DOMAIN added to the INTERESTED IN ROLE statement narrows the client's interests to the specified role and all its subroles that are defined for the specified domain. For example, the statement INTERESTED IN ROLE TOP WITH DOMAIN *Meat* means that the client is interested in all pairs of individuals which are related to each other with the role *canBeMixedWith* or *canBeStuffedWith*. There is only one pair (*i5*, *i1*) related with the role *canBeStuffedWith*.

The keyword WITH RANGE added to the INTERESTED IN ROLE statement narrows the client's interest to the specified role and all its subroles that are defined for the specified range. For example, the statement INTERESTED IN ROLE TOP WITH RANGE *Fruit* means that the client is interested in all pairs of individuals which are related to each other with the role *canBeStuffedWith*. There is only one such pair (*i5*, *i1*).

3.1.3. Interested in Attribute

The statement INTERESTED IN ATTRIBUTE is used by a client to specify its interest in values of the specified attribute. The statement has one required parameter *attribute_expression*, which defines the attribute whose values the client is interested in. For example, the statement INTERESTED IN ATTRIBUTE *hasName* means that the client is interested in names of individuals from the domain of interest.

The statement can appear with additional keywords: FOR, WITH DOMAIN and WITH RANGE. Keywords WITH DOMAIN and WITH RANGE narrows the domain of interest analogically as for INTERESTED IN ROLE statement. The keywords FOR narrows the domain of interest in the way described below.

The keyword FOR narrows the domain of interest to the specified attribute for the set of individuals belonging to the specified concept. For example, the statement INTERESTED IN ATTRIBUTE *hasVitamin* FOR *Fruit* means that the client is interested in the values of the *hasVitamin* attribute only for those individuals which are or may be fruit (depending on the previously defined the area of interests). In this case, the client is interested in the values of *hasVitamin* attribute for individuals *i1* and *i2*.

3.2. Data View Materialization

After the manifest has been specified, the part of knowledge it embraces can be transformed (closed) and encoded as a relational database. We provide here rules for transforming knowledge into entity relationship diagrams (ERD). The rules are as follows:

- Each concept specified in the manifest as interesting one is transformed into a single entity set.
- Such entity set contains all attributes specified as interesting ones for the represented concept and concepts that are placed higher in the hierarchy (see the last point).
- Each role between concepts is transformed into a relationship between entity sets representing these concepts; each relationship is of many-to-many multiplicity.
- Each subsumption (i.e. inheritance) relationship between represented concepts is transformed into IS_A relationship. The set of IS_A relations corresponds to the hierarchy of the represented concepts in an ontology.

ERD created in this way represents the logical model of the closure. The way it is transformed into a database schema is implementation-specific.

The further course of action is dependent on the capabilities of the client component. The client component may require a specific database schema or may be able to operate with the schema generated on the basis of the manifest. In the former case an additional step is needed that creates SQL views translating the generated schema into a client-specific schema. In the latter case no further refinements are needed as the client can adapt itself to data it receives (e.g. in the case when it is developed concurrently with knowledge-based components).

In the next section we present a case study of how to define a manifest describing needs for knowledge for an exemplary ontology and how to transform that ontology into ERD.

3.3. Case Study

In this section we demonstrate a simple case study showing an exemplary way to use the presented Data View approach. Let us assume that we have a system with a knowledge-based component. The component processes a nutritional ontology whose fragment is presented in Figure 3 (in fact this is an excerpt from the real PIPS ontology slightly adapted for the sake of this example).

Let us further assume that we want to integrate the system with some existing already developed simple recipe-holding component. The component can offer its users some aid in the process of creation a recipe by providing them with a list of known products divided into several subcategories. The component can also occasionally give its users some additional information about products they use.

On the basis of a technical specification of the data-based component or by contacting its authors a software engineer is able to prepare a manifest identifying areas of knowledge that might be exploited by the component. An exemplary manifest is presented in Figure 4.

```

INTERESTED IN CONCEPT HIERARCHY POTENTIALLY Food
NOT INTERESTED IN CONCEPT HIERARCHY Fruit
INTERESTED IN CONCEPT Fruit
NOT INTERESTED IN ROLE NOT Fruit canBeServedWith TOP
NOT INTERESTED IN ROLE TOP WITH DOMAIN DESCENDANTS (Food)
NOT INTERESTED IN ATTRIBUTE TOP
INTERESTED IN ATTRIBUTE hasName
INTERESTED IN ATTRIBUTE hasVitamin FOR Meat

```

Figure 4. A manifest of a recipe-holding component

Having in mind that any succeeding statement may cancel the effect of a preceding one, we can interpret the manifest as follows: The first statement declares that the component can take advantage of information about anything that is possibly (POTENTIALLY) edible (is at least potentially a member of the concept *Food*) and about subdivision of such individuals into categories (HIERARCHY).

The next two statements declare that the component is not interested in breakdown of fruits into subcategories (the first statement), but is interested in information which individuals are fruits (the second statement). In the further part of the manifest we read that the component is not interested in the role *canBeServedWith* unless the role subject is a member of the concept *Fruit*. We should neglect all roles that are defined for *Food* subconcepts. The set of attributes the component is interested in embraces *hasName* (for all foods) and *hasVitamin* (but only for meats).

Using methods presented in the previous section the manifest can be transformed into a data model presented in Figure 5. On the basis of this ERD model a relational database can be created and populated with data by the knowledge-based component.

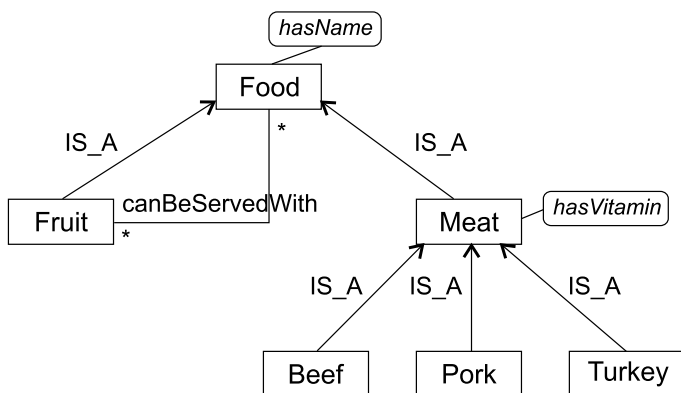


Figure 5. Data model generated on the basis of the manifest from Figure 4

The knowledge-based component can be scheduled to update the world closure, e.g. on a daily basis. The method of communication between components can be further refined by introducing SQL views of the created relational database. In any case the “traditional” component may be completely unaware of existence of an ontology or knowledge processors.

4. Summary

This paper summarizes the experience of authors gathered during development of a knowledge-based system and provides software engineers with a convenient toolbox that can be exploited for overall knowledge-related system design. A novel manifest language can be helpful in describing areas of interest of involved parties and design of interfaces and processes involved in inter-component communication. The language is simple enough to be included as a tool in the process of gathering and evaluation of system requirements. Created manifests can be helpful for design of a system-level ontology. Alternatively we can assume that a dictionary of clients' vocabulary is created in the form of an ontology during requirements gathering phase (a reasonable assumption for a knowledge-based system), making a step towards knowledge-centred requirement management.

Techniques presented here can be further developed and tailored to needs of specific projects. Particularly, knowledge acquiring techniques can be applied to Data Views making the communication between OWA and CWA components bidirectional (this issue has been intentionally neglected in this paper because the question of collecting knowledge from relational databases has been widely discussed e.g. in [8], [9]).

Knowledge engineering is dynamically evolving. The importance of problem of designing good knowledge-based application is expected to increase in the following years. Software engineering should keep pace with this developing technology in order to provide methods of proper introduction of knowledge processors into existing and newly created systems.

Acknowledgements

The authors would like to express their gratitude to participants of the PIPS project taking part in integration of Knowledge Management System with other components of the PIPS system. Special thanks should be given to Riccardo Serafin from foundation San Raffaele del Monte Tabor (Milan) for his analysis of requirements put on Knowledge Management System.

References

- [1] Goczyła K., Grabowska T., Waloszek W., Zawadzki M.: Issues related to Knowledge Management in E-health Systems, (in Polish). In: *"Software Engineering – New Challenges"*, Eds. J. Górski, A. Wardziński, *WNT 2004*, Chap. XXVI, pp. 358-371.
- [2] Goczyła K., Grabowska T., Waloszek W., Zawadzki M.: Inference Mechanisms for Knowledge Management System in E-health Environment. Submitted to the *VII Polish Software Engineering Conference 2005*.
- [3] Staab S., Studer R.: Handbook on Ontologies, *Springer-Verlag*, Berlin, 2004.
- [4] Holsapple C. W.: Handbook on Knowledge Management, *Springer-Verlag*, Berlin, 2004.
- [5] Baader F.A., McGuinness D.L., Nardi D., Patel-Schneider P.F.: The Description Logic Handbook: Theory, implementation, and applications, *Cambridge University Press*, 2003.
- [6] Bechhofer S.: The DIG Description Logic Interface: DIG/1.1, University of Manchester, February 7, 2003.
- [7] DIGUT Interface Version 1.3. KMG@GUT Technical Report, 2005, available at http://km.pg.gda.pl/km/digut/1.3/DIGUT_Interface_1.3.pdf

- [8] Levy A.Y.: Logic-based techniques in data integration, Kluwer Publishers, 2000.
- [9] Arens Y., Chee C., Hsu C., Knoblock C.A.: Retrieving and integrating data from multiple information sources, *International Journal of Intelligent & Cooperative Information Systems*, 1993.
- [10] NeeK Language Grammar Specification in BNF, version 1.0, file available at <http://km.pg.gda.pl/km/NeeK/1.0/Grammar.txt>

The Specifics of Dedicated Data Warehouse Solutions¹

Anna ZYGMUNT, Marek A. VALENTA and Tomasz KMIECIK
*AGH University of Science and Technology, Department of Computer Science,
Al. Mickiewicza 30, 30-059 Kraków, Poland*
e-mails: {azygmunt, valenta}@agh.edu.pl, kmiecik@student.uci.agh.edu.pl

Abstract. Data warehouses are currently one of the most dynamic areas of database development. Knowledge in this area is valuable for the developers of applications, especially when it concerns a rarely touched-upon domain. The authors would like to present their experience with the practical implementation of a data warehouse which stores and processes data related to the incidence of nosocomial infections in hospitals. The paper presents the methodology of creating a conceptual data warehouse model as well as the resultant system architecture. We focus on aspects of managing the data warehouse itself and propose methods for creating and updating the data warehouse in light of the specific conditions posed by the application domain. In order to manage our data warehouse we have created a dedicated application, whose development process is also described in this paper.

Introduction

Data warehouses (and the associated OLAP processing methodology) were initially meant as databases created and implemented for the specific purpose of decision support. Hence, their intended end users included higher-level management, analysts, corporate authorities and all persons in charge of strategic decision making. The goal of the data warehouse was to aid users in the execution of their managerial duties. However, experience dictates that in real life the number of strategic decisions undertaken by management is quite limited and that data warehouses can instead be applied as a highly efficient tool enabling the creation of ad-hoc multidimensional reports and as a source of integrating high-quality data, used in subsequent analysis and knowledge discovery processes (i.e. data mining technologies and tools) [1,4]. The concepts of data warehouses and data mining are key elements of the recently-developed Business Intelligence domain. Unfortunately, the process of constructing and managing a data warehouse is a highly complex endeavor [5]. The first difficulty lies in determining a correct conceptual model of the warehouse. Such a model should be based on a requirements analysis involving future users as well as on an analysis of data being processed by the organization in question. The selection of tools for implementing (and later managing) the actual system is also of profound importance to the success of the whole project. And yet, selecting proper data analysis tools is not – in the authors' opinion – the only prerequisite of success. We believe that it is far more important to

¹ This work is partially sponsored by State Committee for Scientific Research (KBN) grants no. 4T11C02323.

accurately determine the set of tools which will later be used for back-end data warehouse management. The lifecycle of the data warehouse consists of the first major upload of data and of subsequent cyclical updates. The paper presents selected problems which relate to the process of creating and managing a data warehouse, update methodologies and a client application which is used to manage a functional warehouse. The described concepts have been implemented using Microsoft data warehouse development tools: MS SQL Server 2000 and MS Analysis Services. The warehouse stores and processes data related to the incidence of nosocomial infections in hospitals and has been developed with the aid of the Polish Committee for Scientific Research grant 4T11C 023 23, in cooperation with physicians and microbiologists.

1. The Methodology of Data Warehouse Development

The presented data warehouse has been developed in cooperation with the Chair of Microbiology of the JU Medical College and the Polish Society for Nosocomial infections (PTZS), and it furthers the research conducted by both of these organizations. Both institutions have developed a program of active registration of nosocomial infections, basing on the general assumption that such infections are quite natural and unavoidable, but that their frequency can and should be controlled [2]. The problem domain is very wide and hence decision support may potentially relate to many of its aspects – for instance the selection of treatment processes, support for organizational decisions as well as effective management of healthcare funding in light of the dangers posed by nosocomial infections.

The registration of infections proceeds in a uniform way, basing on a set of diagnostic criteria, registration cards and a special application developed by PTZS. The program involves 25 hospitals of varying sizes and various degrees of specialization. Over a two-year period (2002–2003) the database was filled with some 50,000 descriptions of interventional procedures and around 1000 cases of surgical site infections.

The analysis of data thus gathered (for each year separately) is conducted in a centralized manner by PTZS, typically through importing subsets of the database into spreadsheets or by direct SQL queries submitted to local database units. Nevertheless, the growing demand for complex, multidimensional analyses of the complete data set have caused this mode of operation to become obsolete. Hence, the decision was taken to develop a data warehouse.

The following sections present stages of designing a data warehouse (i.e. creating a conceptual model), according to the widely-accepted methodology [6] which consists of four steps: selecting the business process, declaring the grain, choosing the dimensions and identifying measures.

The first step in designing a data warehouse is to decide on the goal which it should fulfill. This decision should be preceded by detailed analysis of organizational processes as well as by attempts at understanding the data used by the organization. In the described case, analyses of data and discussions with physicians have led to the conclusion that medical research programs involve various types of data relating to surgical site infections in patients being treated at Polish hospitals, which participate in the PTZS program on Active Supervision of Nosocomial infections.

Physicians are particularly interested in analyses which would lead to better understanding of the incidence coefficient, defined as the ratio of surgical site infections to the number of surgeries conducted. Following detailed analysis, it was determined that

the gathered data would in fact permit determining this ratio for individual hospital departments, depending on the type of infection and the type of surgical procedure. In addition, requests were made for analyses of the impact on this ratio exerted by selected risk factors, the mode of operation and the type of the administered antibiotic treatment. The need to perform such analyses in a quick and flexible manner was thus established as the first and foremost goal of the data warehouse.

Furthermore, physicians also proved interested in analysing the outcome of the conducted microbiological studies, including (as the most interesting aspect) the analysis of dominant causes of surgical site infections depending on the mode of operation, the cleanliness of the area of operation and the type of infection. Even though only a relatively small percentage of infections are subject to microbiological study, the decision was made to include such data in the warehouse as its secondary goal.

The next step of creating a conceptual data warehouse model is to decide on the level of detail of data stored in the system. It is recommended that the data be stored at the lowest possible level of detail (even though they are rarely presented to the user in this form). Such data are valuable due to their high analytical flexibility: they can be constrained and rolled up in many distinct ways. When designing a data warehouse it is difficult to foresee all possible modes of analysis – hence, data should always be available for ad-hoc queries for which no predetermined aggregations can exist.

In our case, the most atomic piece of data involves a single patient subject to a single surgical procedure (including patients who fall victim to nosocomial infections).

Basing on the analysis of the problem domain conducted in the first step of the described process, we have decided to utilize star- and snowflake-type dimensions. The latter group of dimensions includes *time*, whose hierarchy consists of various levels: quarters, months and days (each level has its own dedicated table). A similar structure is applied to the *hospital department* dimension (each department belong to a given group) as well as to *patient age* (each patient belongs to a given age group). The star-structured dimensions (i.e. ones based on a single table) include: *antibiotics*, *hospitalization time*, *etiological factors*, *surgical area cleanliness*, *operation duration*, *operation mode*, *analyzed material type*, *operation type*, *infection type*, *infection determination*, *state according to ASA*, *procedure mode*, *coinfections*. Each of these dimensions is associated with a set of attributes which constitute its internal hierarchy.

The last step involves determining measures for individual facts. In our case the numerical values subject to analysis (in the dimensions mentioned above) include: *number of operations*, *number of infections* (occurring during the operations) and *infection ratio*. The two initial measures are additive with respect to all dimensions; the third one isn't. Hence, the values for the two initial facts can be stored in a table, while the third value must be calculated for each analysis, basing on the number of infections and the number of operations conducted.

In addition, it turns out that the *number of infections* measure can be analyzed in two distinct ways, depending on the given goal (both types of goals having been described above). Thus, the number of infections may be dependent on the type of infection, surgical area cleanliness, operation duration, operation type and type of anesthesia (first class of analyses), but it can also depend on the determined infection, etiological factors, type of analyzed material and the administered antibiotics (second class of analyses). These considerations have led to the conclusion that our data warehouse should be modeled as a constellation of facts. We have created two fact tables for both classes of analyses. Some of the dimensions relate only to the first class (i.e. *operation*

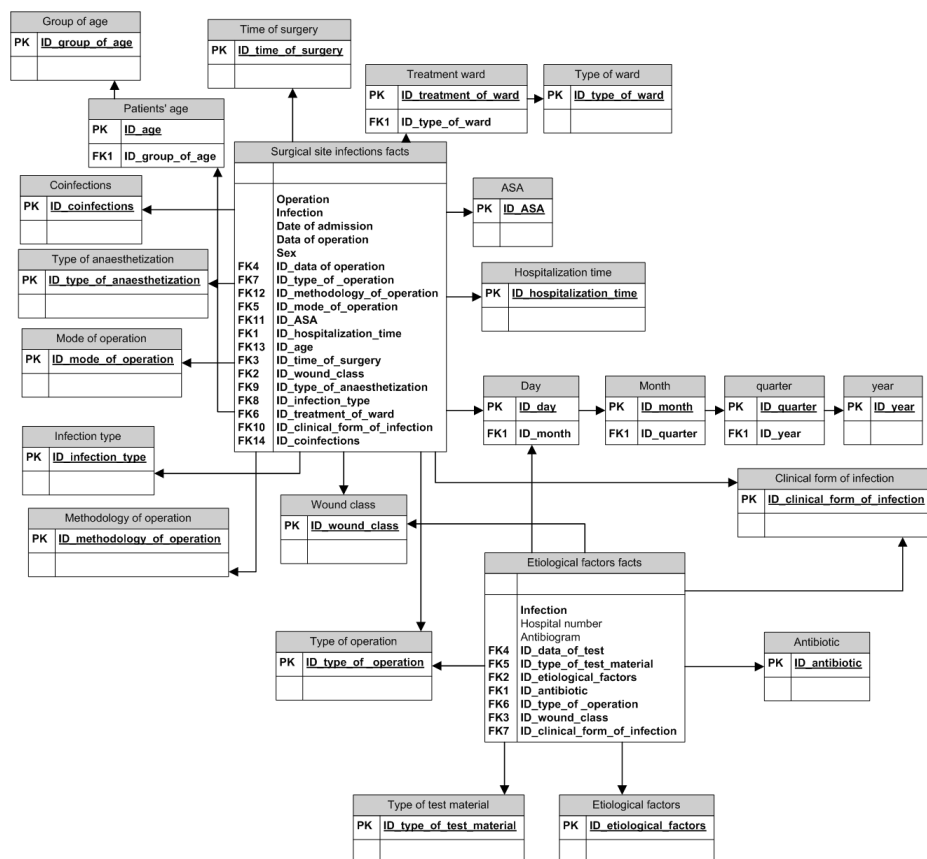


Figure 1. Conceptual data warehouse model.

duration), some only relate to the second one (i.e. *analyzed material type*) and some are common to both classes (i.e. *infection determination*). Figure 1 presents the full model of the data warehouse (in order to preserve readability we did not include individual attributes of each dimension).

2. Data Warehouse Architecture

The architecture of the implemented data warehouse consists of the following components (Fig. 2):

- Source data – data concerning individual cases of nosocomial infections as well as some of the dictionaries are stored in an MS Access database; the remaining dictionaries are presented as text files.
- Staging area – a relational database is used to store new, unprocessed data coming from various sources.
- Subject area – a relational database stores processed data and is used as a source for the multidimensional database.



Figure 2. Data warehouse architecture.

- DTS packages contain the entire logic of ETL (Extract-Transform-Load) processes.
- Multidimensional database.

2.1. The Role of Staging and Subject Databases

The subject database stores processed, integrated data and is used as a source of data cubes. The dependencies between fact tables and dimensions tables are implemented on the basis of rapid and relatively short surrogate keys.

The staging database is used in support of ETL processes. This base enables extracting data from sources as well as their proper processing and removal of inconsistencies. Its structure is similar to that of the subject database, but it uses longer business keys (much like the source databases) and has no explicit relations defined between tables. All data in this database are removed prior to each update of the data warehouse, reducing its complexity and accelerating the process of data extraction and processing.

Even though both databases seem similar in structure, each one performs a different role. The subject database is used solely as a source of data for cubes, hence it can only contain processed data and it must be optimized for query processing. This database cannot contain natural business keys joining fact tables and dimension tables – such keys are often implemented as character strings and preserving them would almost certainly impact the efficiency of the entire data warehouse, causing drill-through operations to become impractical. Furthermore, the subject database must contain constraints ensuring data integrity. These constraints only affect the performance of insert and update operations, which occur relatively rarely and thus aren't considered critical.

The staging database is used solely for ETL processes and it contains unprocessed data. The most common types of operations in the staging database include insertions, updates and deletions. These operations typically affect single fact tables or dimension tables; hence defining constraints and surrogate tables in this database is unnecessary.

2.2. Data Transformation Service (DTS) Packages

DTS packages contain the entire logic of ETL processes. For the purposes of the presented data warehouse, these packages have been divided into two groups:

- packages responsible for loading the staging database
- packages responsible for loading the subject database

Packages of the former type are relatively simple – their only task is to copy all data from the staging database and assign the proper surrogate keys, used to define relationships between fact and dimension tables.

The latter group of packages is far more complex. Such packages are used to load, integrate and process data coming in from various sources (such as MS Access and text files) as well as to store them in fact and dimension tables.

Packages are organized in a hierarchical manner. The main package, called *Import data*, is responsible for the whole ETL process. This package executes two other packages. The first one of them – *Load staging* – uploads data to the staging database, while the second one – *Upload subject* – moves the data to the subject database. Each of these packages is only responsible for uploading fact tables – dimensions are loaded and updated by two other packages: *Load dimensions* and *Update dimensions*.

All packages use properties that are not known at design time and must be dynamically set during execution – such as names of source files, location of other packages and address of servers where staging and subject databases are installed. Achieving this functionality is possible by using DTS Global Variables, which are applied by every package at the beginning of execution to initialize their dynamic properties.

Because global variables can be easily passed between packages and their values can be set by external applications, they are especially convenient for our case. The end-user application, described later in this paper, ascribes values to appropriate variables only once, when executing the *Import data* package, and these values are later passed to all other packages. This ensures that all packages contain consistent information.

2.3. The Multidimensional Database

The data warehouse, created through the use of MS Analysis Services, is a virtual entity. It consists of cubes and their shared dimensions. Cubes are stored in the MOLAP technology: each multidimensional table contains copies of simple data (read in from the subject database) as well as preprocessed aggregations. In the framework of our project – considering the goals described in Section 1 – two cubes have been constructed: *Surgical site infections* and *Dominant etiological factors*. Both cubes rely on the subject database for actual data.

3. Incremental Updates

The lifecycle of the data warehouse consists of an initial upload of data from sources and then of a series of cyclical updates, depending on the modifications of source data. Updating the data warehouse requires recalculation of all aggregates whenever the underlying data change. For dynamically-changing data sources it is necessary to seek out a compromise solution, weighting the necessity of acquiring the most up-to-date information against the effort required to perform the update processes. In the described nosocomial infection data warehouse the situation is relatively simple: hospitals gather data in their local databases, then, at the end of each calendar year, they are obliged to submit all data to the PTZS. At this point the data warehouse is updated through a spe-

cialized management application. This section describes the concept of incremental data warehouse updates, while the next section will focus on implementation aspects.

In simple cases there is a single column (for example an identity or date column) in one of the source tables that can inform us which rows have been added to the fact table since the last warehouse update. In our project the situation is slightly more complicated: the data source is an Access database file, which is replaced by a new file every year. The new file doesn't follow an identical column enumeration pattern, but instead starts a new one from scratch, so we are unable to rely on this mechanism. The date column is also useless, because there is no guarantee that rows are inserted into database in an ascending date order.

We have solved this problem by using two pieces of information to locate rows that have already been loaded: the id number of the last loaded row (separate for each table used as a data source for one of the fact tables) and the last data source file name, both stored in the settings table. First, we read in the last data source file name and compare it with the current source file name. If these names are different, we set the value of last loaded id number to -1, which is always less than any of the existing numbers; otherwise, we simply read in the value from the settings table. Next, while retrieving data from the source database, we load only the rows with the id number greater than the last id number. At the end of executing the DTS package we store new information about the file name and the maximal id number in the settings table.

3.1. Changing Dimensions

In most cases it is possible for dimensions in the data warehouse to undergo changes during the update process. Ralph Kimball [6] named them "slowly changing dimensions" and defined three types thereof.

In type one, used in almost all cases in our data warehouse, the existing row in the dimension table is replaced with new values. The main advantage of this approach is that it is very simple to implement – all we have to do is compare the new row read in from the staging database with the existing row, stored in the subject matter database, and then replace the existing row in case they are different (this is done by using the *update* command; no new row is inserted). The disadvantage is that we do not store the history of changes and hence lose information about old values. This approach is especially convenient when changes are corrections and we are not interested in storing old, invalid data.

In type two (slowly-changing dimension) we insert a new row for every detected change. Contrary to type one, type two preserves historic data. This type is not much more difficult to implement, but we must carefully check if a change is more than just a correction that shouldn't produce a new row. In our data warehouse we use this type only once, for the dimension that stores hospital departments. These departments may occasionally merge or split, and thus simple replacement of an existing row in the dimension table is not possible.

In type three (slowly-changing dimension), which we don't use in our data warehouse, every change is written to the same row, requiring multiple columns that store the same attribute of a dimension. The disadvantage of this approach is that it has limited capacity for storing historic data and is more difficult to implement.

For every type of changing dimension there is a need to detect which rows were changed. The conceptually simplest approach is to compare rows with the same business key from the staging database (which stores new data) and the subject database,

column by column. We proceed in a slightly different way – for every row in the dimension table we compute a hash value (using the *checksum* or *binary_checksum* SQL functions), then store this value in one of the columns. During update, we compute hashes for every new row and compare them with values stored in the appropriate row (i.e. the row with the same business key) in the dimension table to find out if it should be changed.

4. Installation and Configuration

Usually an organization includes a separate administrator role, responsible for installing and configuring the whole data warehouse. However, our case is different: the data warehouse will only be used by several people with little or even no database experience, and the process of installing, updating and configuring should be as simple as possible. Users won't be able to manually execute DTS packages and run SQL scripts, but instead expect to manage the warehouse in the same way they use other Windows application. These requirements are a challenge for warehouse designers, because standard tools are typically designed for experienced database operators. This section describes our approach to the problem.

4.1. The End-User Application for Managing the Data Warehouse

As mentioned before, in our case updating data warehouse should be as simple as possible. To this end, we have created a small application that allows users to enter the required information and update the data warehouse using a single button. This application also helps users enter appropriate values by displaying short help messages and checking the entered values for consistency.

The settings entered by user are stored in an XML file and we access them using the configuration application block, which is part of the Microsoft Enterprise Library [3,7]. Application blocks are reusable components designed to assist in common enterprise development tasks. A configuration application block simplifies access to data stored in various data sources. Among others, using application blocks instead of writing own components for accessing configuration data has the following advantages:

- The application block is well tested.
- Settings don't have to be only string values that must be manually parsed in our application, but can be of any serializable type.
- It is very simply to use: reading and writing settings data relies on single lines of code.
- It can be easily configured using the Microsoft Configuration Console.

The behavior of application blocks can be easily changed by implementing various *providers*, which are interfaces describing the most fundamental functions that an application block uses to perform as intended. The application configuration block, like every other element of the Microsoft Enterprise Library, already contains useful implementations of providers, which in this case include the *Storage Provider* and the *Transformer*.

The *Storage Provider* is used to read data from a data source. In its default implementation (*XmlFileStorageProvider* class) the data source is an XML file, al-

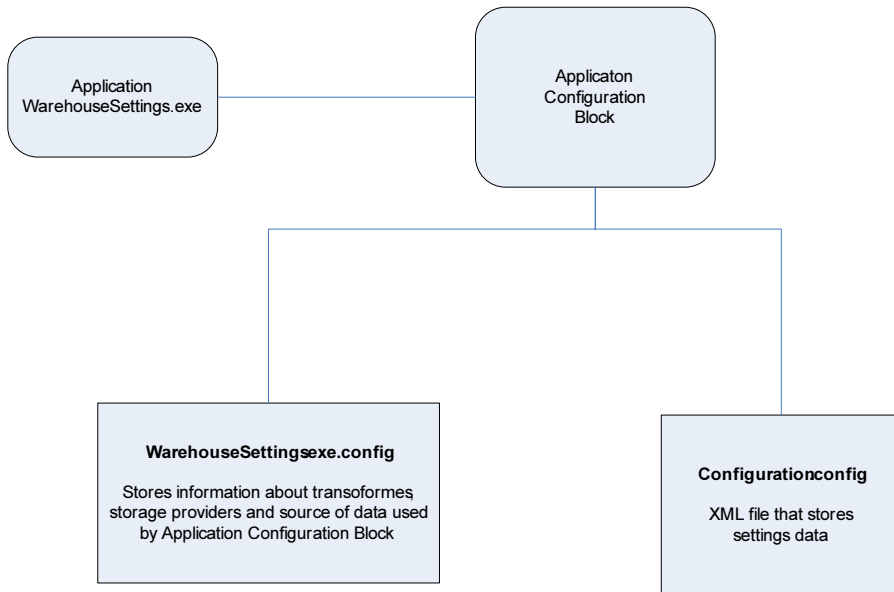


Figure 3. Using the Configuration Application Block.

though this can be replaced by another class that can read data from a database, an INI file, the Windows Registry or any other source that required by the application.

The *Transformer* is used to transform data from the representation that is returned by the *Storage Provider* to the requested class, and to transform the user class that describes data to a format acceptable by the data source. The implementation included in the Enterprise Library (`XmlSerializerTransformer` class) converts XML nodes to any serializable type defined by the developer.

The *Transformer* provider is optional, and if not used, the data returned to user is of the same type as the data returned by the storage provider.

In our application we use default provider implementations and store configuration data in an XML file. Figure 3 presents how this system is used.

The `WarehouseSettings.exe.config` file is a default XML configuration file for Windows applications and it is used by the Configuration Application Block for storing information about used providers and sources of data (in our case – a filename). The `Configuration.config` file contains settings required by the application.

Changing default providers can be done without changing application code, and it only requires two steps. First, we have to implement a provider and place it in an assembly, that can be copied to the application folder or installed in the Global Assembly Cache. Second, we have to change the `WarehouseSettings.exe.config` file to point to appropriate classes that can be used to read in and transform data, and select the new data source. Although the second step can be done manually, the recommended way is to use the Enterprise Library Configuration application included in the Microsoft Enterprise Library, which not only creates valid XML output, but also can check the entered values for consistency.

4.2. Installation

The simplest way to install the application is just to copy all the required files to the desired destination (this is sometimes called XCOPY deployment). Such an approach is possible for small projects, but cannot be used in our case, because apart from copying all DTS packages, data source files and the end-user application, the installer must also perform the following operations:

- Check if all requirements on the target machine are met. These requirements include .NET Framework 2.0, Microsoft SQL Server and Microsoft Analysis Services.
- Create the staging and subject databases and fill the time dimension.
- Create the multidimensional database.
- Write appropriate initial configuration values.

All these operations can be done by using Microsoft Windows Installer, an installation service that is part of the Windows operation system. It offers many features that are needed in our case:

- It allows to specify conditions that must be met to install an application. We search the Windows registry to check if the SQL Server and Analysis Services are installed.
- It can execute custom code during installation. We use this feature to create the staging, subject and multidimensional databases, and to write appropriate values to the end-user application configuration file.
- The installation is processed as a single transaction. If the installation of any part of the application fails, it is rolled back and all previously-installed components are uninstalled.
- It allows uninstalling and fixing the application.

The Windows Installer Package can be created manually using the Windows Installer Software Development Kit, but we have used the Microsoft Visual Studio 2005 .NET, which allows building setup projects using a visual designer and supports custom code written in any language that is compatible with the .NET Common Language Specification.

As mentioned before, the Windows Installer enables the execution of custom actions. These actions can be presented as any executable file or an assembly that stores an Installer class. The Installer class (its full name in the .NET Framework Class Library is `System.Configuration.Install`) contains four virtual methods: *Install*, *Commit*, *Rollback* and *Uninstall*. The first three of them are used during installation, and the fourth is used when an application is being uninstalled. We have created our own installers by extending the Installer class and overriding these methods.

5. Conclusions

The paper presents selected aspects of creating and managing a data warehouse, with a practical implementation in the medical subject domain. The specifics of the implemented warehouse are a consequence of the goals determined by the medical researchers who will make use of the system.

By presenting our approach, we wish to address specific details of the problem as well as the selection of appropriate solutions. The paper describes a holistic approach to the design and implementation of a data warehouse, not just in the context of its immediate goals (analyses of nosocomial infections), but also considering its future users and administrators. Hence, the typical set of data warehouse requirements, which includes rapid and easy access to the required analyses, has been extended with the need to create an environment for user-friendly management of the system, well suited to the needs of medical researchers. When considering the conceptual layer of the project, we also determined that it should support analyses enabling the verification of heuristic knowledge possessed by epidemiologists involved in the database design process. This knowledge can be extended and supplemented through data mining techniques, focusing on the subject database, which contains integrated and preprocessed data – an important issue for planning the use of nosocomial infection knowledge models in modern decision support systems (for instance in expert systems [8,9]).

References

- [1] Connolly T., Begg C.: Database Systems: A Practical Approach to Design, Implementation and Management., Pearson Education Limited, 2002.
- [2] Damani N.N.: Praktyczne metody kontroli zakażeń – Manual of Infection Control Procedures. *Polskie Towarzystwo Zakażeń Szpitalnych*, Kraków, 1999.
- [3] Database Journal: <http://www.databasejournal.com>.
- [4] Han J., Kamber M.: Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2001.
- [5] Jarke M., Lenzerini M., Vassiliou Y., Vassiliadis P.: Fundamentals of Data Warehouses., *Springer-Verlag*, 2000.
- [6] Kimball R., Ross M.: The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling., *John Wiley & Sons*, 2002.
- [7] Microsoft SQL Server Accelerator For Business Intelligence, <http://www.microsoft.com/sql/ssabi/default.asp>.
- [8] Valenta M.A., Zygmunt A.: Process of building the probabilistic knowledge model, in.: Selected problems of IT application, ed. Grabara J.K. – Warszawa: *Wydawnictwa Naukowo-Techniczne*, 2004.
- [9] Valenta M.A., Zygmunt A.: Probabilistic medical knowledge and its application in expert systems in.: *Inżynieria wiedzy i systemy ekspertowe*. T. 1 ed. Bubnicki Z., Grzech A., Wrocław: *Oficyna Wydawnicza Politechniki Wrocławskiej*.

Formal Approach to Prototyping and Analysis of Modular Rule-Based Systems¹

Marcin SZPYRKA and Grzegorz J. NALEPA

Institute of Automatics,

AGH University of Science and Technology,

Al. Mickiewicza 30, 30-059 Kraków, Poland

e-mails: mszpyrka@agh.edu.pl, gjn@agh.edu.pl

Abstract. The paper discusses a formal approach to prototyping and analysis of modular rule-based systems that are incorporated into embedded systems. To overcome problems appearing during the design of rule-based systems a new approach to the design process, including the new hybrid rule representation method called XTT is presented in the paper. The proposed approach allows to modularize rule-based systems i.e. a complex system can be split into number of modules, represented by a tree-like hierarchy. The approach is supported by computer tools, equipped with verification algorithms that are based on formal methods. It allows for designing reliable rule-based systems that may be incorporated into an RTCP-net model.

Introduction

Computers are used to control a wide range of systems, from simple domestic machines to entire manufacturing plants. The software of these systems is embedded in some larger hardware system and must respond to every stimuli from the system's environment within specified time intervals (see [6]). Multiple real-time systems are control systems that monitor quantities of interest in an environment. In response to changes in the monitored quantities they perform control operations or other externally visible actions. The process of making decision what actions should be performed may be based on a rule-based knowledge base that is incorporated into such an embedded system.

In software engineering (see [6]), the term *formal methods* is used to refer to any activities which rely on mathematical representations of software. The development process of computer systems may be supported by formal methods to ensure more dependable products. In our approach RTCP-nets (Real-Time Coloured Petri nets, see [7]) are used as the modelling language. RTCP-nets are a result of adaptation timed coloured Petri nets (see [1]) to modelling and analysis of embedded systems. RTCP-nets enable to construct a large hierarchical net by composing a number of smaller nets into a larger structure. Moreover, a template mechanism allows users to design models and manipulate their properties in a fast and effective way. RTCP-nets are suitable for modelling embedded systems incorporating an RBS (rule-based system) [2]. Such

¹ Research supported from a KBN Research Project No.: 4 T11C 035 24

a system is represented as a D-net (decision net) that constitutes the bottom layer of the model.

In this paper, we will disregard the time aspects of embedded systems and we will focus on prototyping and analysis of rule-based system that may constitute a part of such embedded systems. The main problem in the RBS design process is that in systems having more than several rules it becomes difficult to preserve their formal properties at the design stage. It is hard to keep the rules consistent, to cover all operation variants and to make the system work according to the desired specification.

These problems may be addressed by using a proper knowledge representation method, which simplifies the design. In these case some classic engineering tools, such as decision tables or decision trees, may be used. However, they tend to have a limited applicability in case of large scale complex systems. This is why special hybrid representation methods combining existing approaches may be developed.

To overcome problems pointed out, an integrated rule-based systems design methodology, including the new hybrid rule representation method is presented in the paper. Design of complex RBSs may be carried out using new visual knowledge representation language called Extended Tabular-Trees (XTT). XTT combine decision-tables and decision-trees by building a special hierarchy of Object-Attribute-Tables (OAT). The approach is supported by computer tools allowing for the incorporation of the verification stage into the design process. During the design stage, users can check whether an RBS satisfies the desired properties. Most importantly the proposed approach allows for *modularization* of the RBS. Presented approach allows for designing a reliable RBS that may be incorporated into an RTCP-net model.

The original contribution of this paper is the idea of combining two formal design and analysis methods for RBSs: XTT, a hybrid design method that allows for D-nets, a Petri Nets-based design method allowing for automatic prototype generation using Ada95. Both methods are supported with CASE tools, that are able to share the RBS design thru XML-base description. XTT is more suitable for hierarchical RBSs, with local per-module verification, while D-Nets are useful in modelling more simple RBS, while providing global formal verification.

The paper is organized as follows: Section 1 deals with a short description of RTCP-nets, especially to indicate the place of an RBS in the model. Selected aspects of RBSs design and verification are presented in Section 2. Section 3 presents Adder D-nets Designer tools that are used to design simple RBSs, represented by single decision table, and to transform them automatically into a part of an RTCP-net. The XTT approach, which allows for design of more complex, modular RBSs is presented in Section 4. The paper ends with concluding remarks and a short summary in the final section.

1. RTCP-nets

RTCP-nets are a formal mathematical construct used for modelling and analysis of embedded systems. They provide a framework for design, specification, validation and verification of such systems. Based upon the experience with application of CP-nets ([1]) for real-time systems' modelling, some modifications were introduced in order to make CP-nets more suitable for this purpose. RTCP-nets differ from timed CP-nets in a few ways.

- The sets of net's nodes (places and transitions) are divided into subsets of main and auxiliary nodes to distinguish a part of a model that represents the structure of a considered systems and its activities.
- In contrast to CP-nets, multiple arcs are not allowed in RTCP-nets. Moreover, for any arc, each evaluation of the arc weight expression must yield a single token belonging to the type that is attached to the corresponding place.
- RTCP-nets are equipped with a new time model. Time stamps are attached to places instead of tokens. Any positive value of a time stamp describes how long a token in the corresponding place will be inaccessible for any transition. A token is accessible for a transition, if the corresponding time stamp is equal to or less than zero. Negative values denote the age of a token. It is possible to specify how old a token should be so that a transition may consume it.
- RTCP-nets use a priority function. Priorities assigned to transitions allow direct modelling deterministic choice.

More detailed description and a formal definition of RTCP-nets may be found in [7]. We will focus on practical aspects connected with the drawing of hierarchical models. For the effective modelling RTCP-nets enable to distribute parts of the net across multiple subnets. The ability to define subnets enables to construct a large hierarchical net by composing a number of smaller nets. Hierarchical RTCP-nets are based on hierarchical CP-nets (see [1]). Substitution transitions and fusion places are used to combine pages but they are a mere designing convenience. The former idea allows refining a transition and its surrounding arcs to a more complex RTCP-net, which usually gives a more precise and detailed description of the activity represented by the substitution transition. A fusion of places allows users to specify a set of places that should be considered as a single one. It means, that they all represent a single conceptual place, but are drawn as separate individual places (e.g. for clarity reasons).

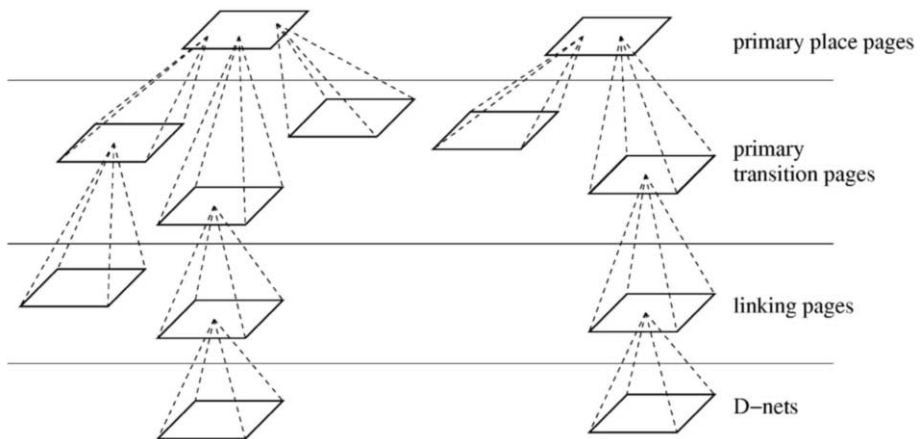


Figure 1. General structure of an RTCP-net in canonical form

There is a special form of hierarchical RTCP-nets called *canonical form*. The canonical form has been worked out to speed up and facilitate the drawing of models and to enable the generation of a model or part of it automatically. RTCP-nets in

canonical form are composed of four types of pages (subnets) with a precisely defined structure:

- *Primary place pages* are used to represent active objects (i.e. objects performing activities) and their activities. They are oriented towards objects presentation and are top level pages. Such a page is composed of one main place that represents the object and one main transition for each object activity.
- *Primary transition pages* are oriented towards activities' presentation and are second level pages. Such a page contains all the places, the values of which are necessary to execute the activity, i.e. the page is composed of one main transition that represents the activity and a few main places.
- *Linking pages* belong to the functional level of a model. They are used (if necessary) to represent an algorithm that describes an activity in details. Moreover, a linking page is used as an interface for gluing the corresponding D-net into a model. Such a page is used to gather all necessary information for the D-net and to distribute the results of the D-net activity.
- *D-nets* are used to represent rule-based systems in a Petri net form. They are utilized to verify an RBS properties and constitute parts of an RTCP-net model. D-nets belong to the bottom level of the model.

The general structure of an RTCP-net in canonical form is shown in Figure 1.

If two or more pages are similar, only one *page template* with a number of parameters may be designed (similar to class templates in C++). A parameter can take any string value. All the similar pages will be generated according to parameter values as instances of the template. Values of the parameters are determined in the hierarchy composition stage. Drawing of all pages or page templates completes the first stage of the RTCP-net design process. Then, some connections among pages must be constituted to compose one hierarchical model. All connections among pages are presented with the use of a page hierarchy graph. Each node represents a simple page, and an arc represents a connection between a subpage and its substitution transition.

2. Rules-Based Systems

Rule-Based Systems are a powerful tool for knowledge specification in design and implementation of knowledge-based systems (KBS). They provide a universal programming approach for domains such as system monitoring, control, and diagnosis.

In most basic version, an RBS for control or decision support consists of a single-layer set of rules and a simple inference engine; it works by selecting and executing a single rule at a time, provided that the preconditions of the rule are satisfied in the current state. The basic form of a production rule is as follows:

rule: IF < preconditions > THEN < conditions > (1)

where < preconditions > is a formula defining when the rule can be applied, and < conditions > is the definition of the effect of applying the rule; it can be a logical formula, a decision or an action.

The expressive power and scope of potential applications combined with modularity make RBSs a general and applicable mechanism. However, despite a vast spread-out in working systems, their theoretical analysis seems to constitute still an open issue with respect to analysis, design methodologies and verification of theoretical properties. Assuring *reliability*, *safety*, *quality* and *efficiency* of rule-based systems require both theoretical insight and development of practical tools.

Rule-based systems should satisfy certain formal requirements, including completeness and consistency. To achieve a reasonable level of efficiency (quality of the knowledge-base) the set of rules must be designed in an appropriate way. For the purpose of this paper the following formal properties are identified:

- *Completeness* – A rule-based system is considered to be *complete* if there exists at least one rule succeeding for any possible input state.
- *Consistency (determinism)* – A set of rules is considered consistent (deterministic) if no two different rules can produce different results for the same input state.

The above definition omits overlapping of rules. A more restrictive definition can be introduced: A set of rules is considered deterministic if no two different rules can succeed for the same state.

- *Optimality (redundancy)* – A set of rules is considered optimal if it does not contain dependent (redundant) rules.

For a complete and consistent set of rules, a dependent rule means a rule that can be eliminated without changing these two properties. The algorithm that searches for dependent rules does not change rules in the considered RBS.

The last considered property allows reducing the number of decision rules in an RBS but usually, only few rules can be eliminated in such a case. To carry out more sophisticated reduction the most general case of subsumption is considered here.

A rule *subsumes* another rule if the following conditions hold: the precondition part of the first rule is more general than the precondition of the subsumed rule, and the conclusion part of the first rule is more specific than the conclusion of the subsumed rule. For intuition, a subsumed rule can be eliminated because it produces weaker results and requires stronger conditions to be satisfied; thus any of such results can be produced with the subsuming rule.

Rule-based systems may be implemented with use of different logical calculi, depending on the expressive power needed. Simple RBSs may use propositional logic, this is the case of Adder Designer. More complex solutions, like XTT, make use of attribute-based language, which combined with classic propositional calculus have better expressive power.

3. Adder D-nets Designer

Computer tools for RTCP-nets, called Adder Tools (Analysis and Design of Embedded Real-time systems), are being developed at AGH University of Science and Technology in Krakow, Poland. Adder Tools contain: Adder D-nets Designer for the design and verification of rule-based systems, Adder Editor for designing RTCP-nets, and Adder Simulator for the simulation of RTCP-nets. *Adder Tools home page*, hosting information about current status of the project, is located at <http://adder.ia.agh.edu.pl>.

We will focus on the first of the tools. The *Adder D-net Designer* consists of: user-friendly interface for RBSs designing, automatic transformation of an RBS into the corresponding D-net, and automatic analysis of D-nets properties. *Adder Designer* uses decision tables as a form of a rule-based system representation.

An RBS is represented by a single decision table. To construct such a decision table, we draw a column for each condition (or stimulus) that will be used in the process of taking a decision. Next, we add columns for each action (or response) that we may want the system to perform (or generate). Then, for every possible combination of values of those conditions a row should be drawn. We fill cells so as to reflect which actions should be performed for each combination of conditions. Each such row is called a *decision rule*. A cell in a decision table may contain a formula, which evaluates to a set of values of the corresponding attribute. Thus, in this approach decision rules should be treated as decision rules' patterns (or generalized decision rules). The number of rules' patterns is significantly less than the number of simple decision rules. In spite of this, we will use the name "decision rules" for decision rules' patterns.

Let's consider a traffic lights control system for crossroads presented in Figure 2. We will focus on designing an RBS for the system.

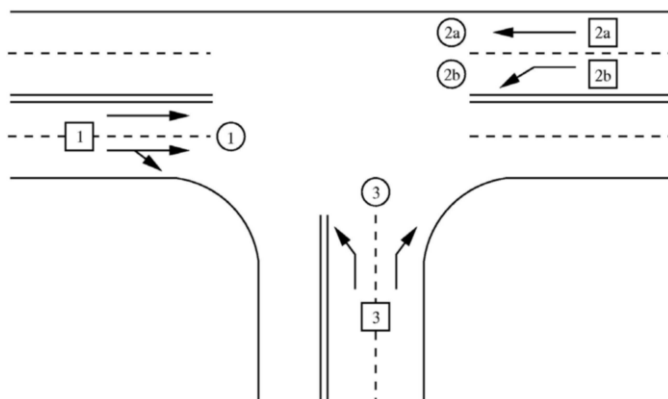


Figure 2. Crossroads model

The system should take into consideration the traffic rate in the input roadways. All roads are monitored, monitors are drawn as rectangles, and four traffic lights are used (drawn as circles). Three different traffic lights' states are possible:

Table 1. Acceptable traffic lights' states

State number	Lights 1	Lights 2a	Lights 2b	Lights 3
1	green	green	red	red
2	red	green	green	red
3	red	red	red	green

The system works in the following way. If there are some vehicles at all input roadways, the states 1, 2 and 3 are displayed sequentially. If there is no need to display

a state, the state is omitted and the next state is displayed. The four monitors are used to determine the state to be displayed next. The RBS (final version) designed for the traffic light’s driver is presented in Table 2.

Table 2. RBS for the traffic lights control system

Conditional attribures					Decision attributes				
S	T1	T2a	T2b	T3	S	L1	L2a	L2b	L3
S = 1	T1	T2a	T2b > 0	T3	2	red	green	green	red
S = 1	T1	T2a	T2b = 0	T3 > 0	3	red	red	red	green
S = 1	T1	T2a	T2b = 0	T3 = 0	1	green	green	red	red
S = 2	T1	T2a	T2b	T3 > 0	3	red	red	red	green
S = 2	T1 > 0	T2a	T2b	T3 = 0	1	green	green	red	red
S = 2	T1 = 0	T2a	T2b	T3 = 0	2	red	green	green	red
S = 3	T1 > 0	T2a	T2b	T3	1	green	green	red	red
S = 3	T1 = 0	T2a > 0	T2b	T3	2	red	green	green	red
S = 3	T1 = 0	T2a	T2b > 0	T3	2	red	green	green	red
S = 3	T1 = 0	T2a = 0	T2b = 0	T3	3	red	red	red	green

The verification stage is included into the design process. At any time, during the verification stage, users can check whether an RBS is complete, consistent and optimal. An example of the *Adder Designer* session is shown in Figure 3. In the window the decision table presented in Table 2 and its completeness analysis results are presented.

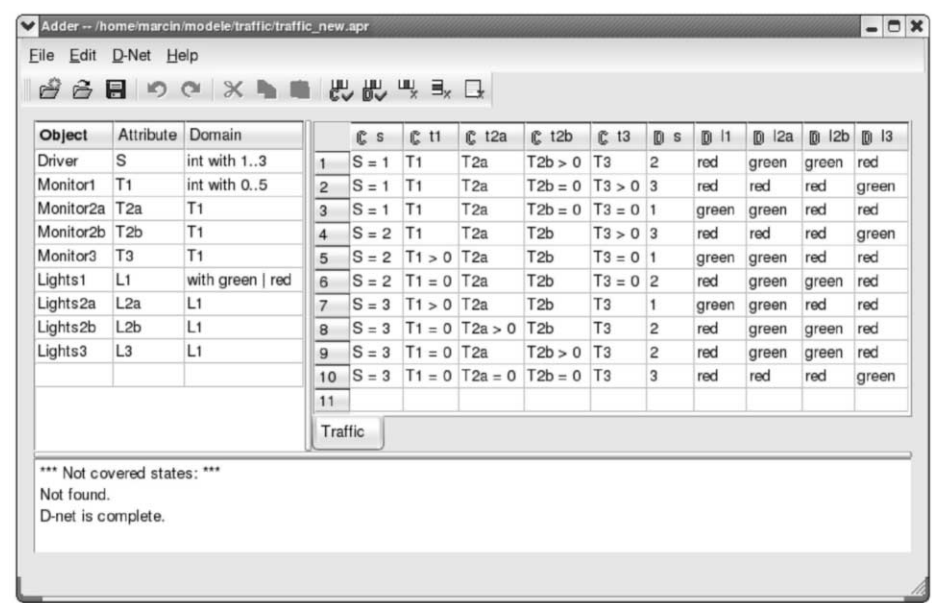


Figure 3. Example of Adder D-nets Designer session

Presented approach allows for designing a reliable RBS that may be incorporated into an RTCP-net model. To be included into an RTCP-net an RBS (presented as a decision table) is transformed into a D-net. A D-net is a non-hierarchical coloured Petri net, which represents a set of decision rules. A D-net form of the decision table presented in Table 2 is shown in Figure 4.

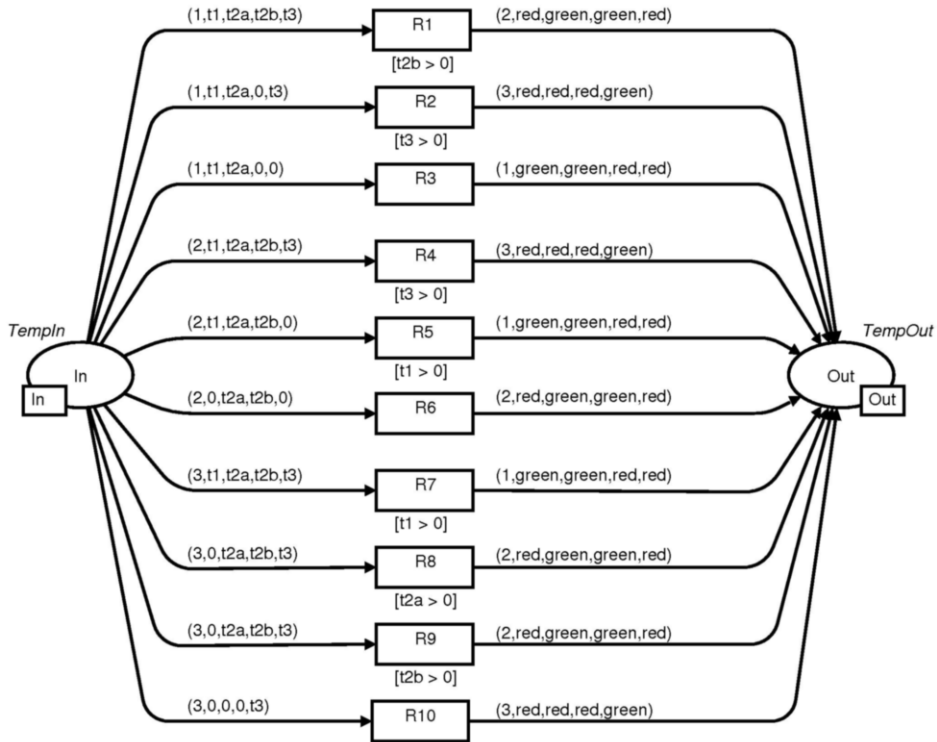


Figure 4. D-net form of the decision table presented in Table 2

A D-net contains two places, *conditional place* for values of conditional attributes and *decision place* for values of decision attributes. Each positive decision rule is represented by a transition and its input and output arcs. A token placed in the place *In* denotes a sequence of values of conditional attributes. Similarly, a token placed in the place *Out* denotes a sequence of values of decision attributes. Each marking of a D-net contains only one token.

Adder Tools are implemented in C++ on the open source platform using GNU/Linux operating system and portable Qt libraries. This platform enables greater code portability, since the source code can be also run on other Qt-supported platforms such as Windows. There is an ongoing effort to formally define a translation of an RTCP-net model to Ada95 code. In this way a prototype preserving all the *formal* aspects of the model could be built.

Major design problems arise in case of non-trivial RBSs that have multiple attributes and many rules. In such cases classic decision table representation is not

optimal, since the table is difficult to build and analyse. This is why proper *modularization* of an RBS is needed. It can be accomplished with use of XTT-based Mirella Designer.

4. Mirella XTT Designer

Mirella is a CASE tool supporting visual design, on-line analysis, and prototyping of rule-based systems. *Mirella* allows for hierarchical design by using system *modularization*. It is oriented towards designing reliable and safe rule-based systems in general. The main goal of the system is to move the design procedure to an abstract, logical and visual level, where knowledge specification is based on the use of abstract rule representation. Internal system structure is represented using new knowledge representation, *eXtended Tabular Trees* (XTT) [5].

The traffic lights control system described in Section 3 has been designed in *Mirella* using the XTT approach. The flat decision table (see Table 2) has been decomposed into a *modular* structure as seen in Figure 5. This has been accomplished by identifying groups of rules with common context (same attribute values), which correspond to XTT table headers.

In this case, the structure consists of five XTT tables corresponding to five different knowledge modules of the control system:

- **state** table describes three possible system states, (it is a root table of the XTT structure),
- **left** table describes the left roadway,
- **right** table describes the right roadway,
- **horizontal** table represents other traffic on the left and right roadways,
- **lights** table shows the system decision, i.e. lights' states.

XTT combines some classic engineering approaches such as decision tables and decision trees. It uses attribute logic and explicit inference information to create special hierarchy of Object-Attribute-Tables [3], [4]. The rows of XTT tables correspond to individual rules. The conclusion part of the rule includes specification of allow/retract attribute values which are added/removed from the rule base during the inference process. This allows for non-monotonic reasoning. Tables can be connected in order to control the inference process.

A formally defined transformation of XTT structure to Prolog-based representation enables both on-line formal system analysis, and automatic prototype generation. During the logical design phase incremental rule-base synthesis is supported by the on-line, Prolog-based *analysis and verification framework*. The framework allows for the implementation of different verification and analysis modules (plugins). System analysis, verification and optimization modules are implemented in Prolog. Each module reads the XTT rule base and performs the analysis of the given property, such as: completeness, determinism, or subsumption. Since the modules have access to the Prolog-based XTT system description, it is also possible to implement dynamic rule correction algorithms.

Using the XTT-based system design a Prolog-based system prototype can be automatically generated. The prototype uses meta-level rule interpretation in Prolog

described in more detail in [5]. Once the Prolog code has been generated it can be tested using the inference engine shell provided with Mirella. Finally a standalone prototype can be generated using SWI-Prolog compiler. In this stage three components are *linked* together: the *XTT inference engine*, the *system rule base*, the *SWI runtime*. This approach allows for building rule-based system applications, running *independently* of Mirella, or SWI-Prolog environment on multiple platforms, including: GNU/Linux, Unix, and Windows.

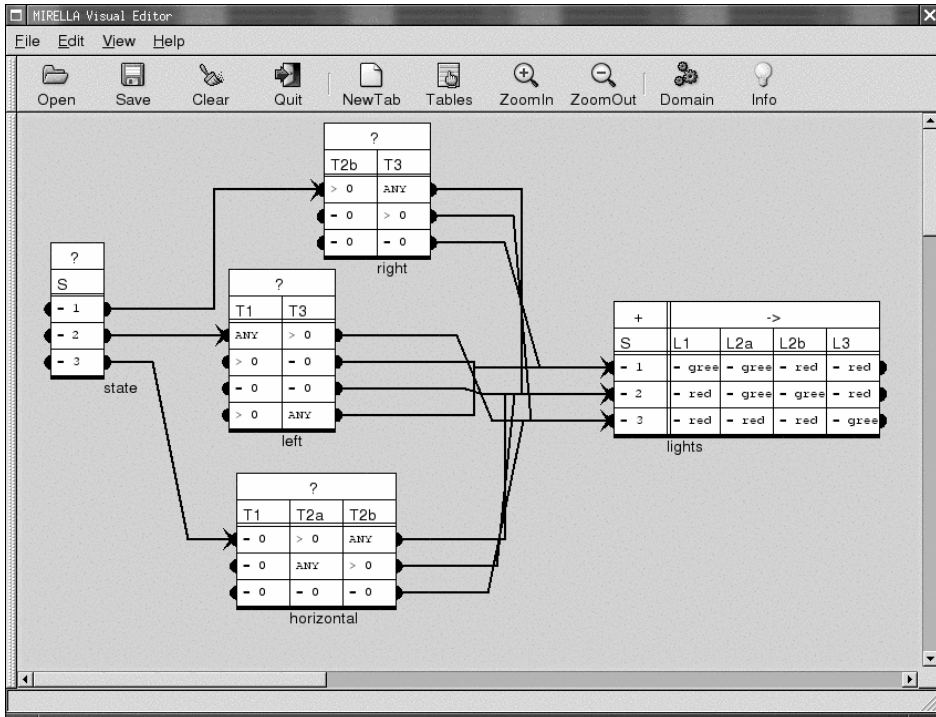


Figure 5. XTT for the traffic lights control system

The visual XTT specification can also be translated into a predefined XML (*XTTML*) knowledge format, so the designer can focus on the logical specification of safety and reliability. XML-based knowledge representation languages are used in order to provide wide data exchange possibilities. *XTTML* provides a direct representation of *XTT* structure. It describes RBS design and can be used to translate *XTT* to other high level knowledge representation formalisms. The XML representation allows for *sharing* the rule base with Adder Designer. They can be used to share the logical RBS structure with other design environments. It is then possible to design an RBS in Mirella, import it to Adder, where it can be analysed, transformed to D-net form and included into an RTCP-net.

Mirella itself aims at supporting an integrated, incremental design and implementation process, support for a wide class of rule-based systems. It has open architecture, allowing for future extensions. The tool itself was developed in “open source” GNU/Linux environment using L/GPL-licensed development tools such as

GCC, GTK, GNOME and SWI-Prolog. It makes the source code portable and allows for multiple operation systems' support. The home page of the project is mirella.ia.agh.edu.pl.

5. Summary

The article presents a formal approach to engineering embedded rule-based systems' design and prototyping. This is a hybrid approach that combines the RTCP-nets (Peri Nets) hierarchical model with logic-based XTT representation. The visual XTT model allows for rule-based system modularization and fast system prototyping thanks to direct translation, to executable, Prolog-based rule representation. Both the RTCP-net-based and XTT-based models enable formal analysis of system properties during the design. The design process is supported by two CASE tools Adder and Mirella exchanging RBS description using an XML-based representation. The tools are implemented using modern open source platform. The approach discussed in the article aims at providing *formal* method to practical design, analysis and implementation of *reliable* and mission-critical rule-based systems, where both real-time constraints and formal properties are of key importance.

References

- [1] Jensen K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 1,2 & 3, Springer-Verlag (1996)
- [2] Liebowitz J. (ed.), The Handbook of Applied Expert Systems, CRC Press, 1998, Boca Raton.
- [3] Ligeza A., Wojnicki I., Nalepa G. J.: Tab-Trees: a CASE tool for design of extended tabular systems. In Heinrich C. Mayr [et al.] (Eds.) *Database and expert systems applications: 12th international conference, DEXA 2001*, Munich, Germany, 2001, LNCS 2113, pp. 422-431.
- [4] Nalepa G. J., Ligeza A.: Designing reliable web security systems using rule-based systems approach. In Menasalvas E., Segovia J., Szczepaniak P. S. (Eds.) *Advances in Web Intelligence: first international Atlantic Web Intelligence Conference AWIC2003*, Madrid, Spain, 2003, LNAI 2663. Subseries of LNCS, pp. 124-133.
- [5] Nalepa G. J.: Meta-Level Approach to Integrated Process of Design and Implementation of Rule-Based Systems, PhD Thesis, AGH University of Science and Technology, Krakow, Poland (2004)
- [6] Sommerville, I.: Software Engineering. Pearson Education Limited (2004)
- [7] Szpyrka, M.: Fast and Flexible Modelling of Real-time Systems with RTCP-nets. Computer Science (2004)
- [8] Szpyrka, M., Szmuc, T., Matyasik, P., Szmuc, W.: *A formal approach to modelling of real-time systems using RTCP-nets*. Foundation of Computing and Decision Science, Vol. 30, No. 1, pp. 61-71 (2005)

Selection and Testing of Reporting Tools for an Internet Cadastre Information System

Dariusz KRÓL, Małgorzata PODYMA and Bogdan TRAWIŃSKI
Wrocław University of Technology, Institute of Applied Informatics,
Wybrzeże S. Wyspiańskiego 27, 50-370 Wrocław, Poland
e-mails: dariusz.krol@pwr.wroc.pl,
podyma@bogart.wroc.biz, trawinski@pwr.wroc.pl

Abstract. The process of selecting and implementing of reporting mechanisms for a cadastre internet information system is described. Many factors as technical feasibility, costs and licensing as well as the efficiency and scalability required have been taken into account. Three versions of reporting mechanisms based on Crystal Reports, Free PDF library and XML technology have been implemented and tested using Microsoft's Web Application Stress Tool.

Introduction

Many articles and books have been written on the methodology of software engineering [4], [9], [12], [14], many deal with requirement engineering [7], [8], and [10]. Software engineering body of knowledge has been developed to express consensus in industry, among professional societies and standards-setting bodies and in academia [1], [15].

Selection of a reporting tool for an internet information system is a difficult task. It should be accomplished carefully and many factors should be taken into account i.e. system functionality, features of system usage, technical feasibility, costs and licensing as well as the performance and scalability required. The process of selecting and implementing of reporting mechanisms in a cadastre internet information system is reported in the paper. During last four years three versions of reporting mechanisms were designed and programmed in the system, i.e. mechanisms based on Crystal Reports [16], Free PDF library [13] and XML technology [11]. The study has been carried out by the Department of Information Systems at Wrocław University of Technology in cooperation with a commercial software company.

1. Structure of the EGB2000-INT Cadastre System

The maintenance of real estate cadastre registers is dispersed in Poland. There are above 400 information centers located by district local self-governments as well as by the municipalities of bigger towns in Poland which exploit different cadastre systems. The EGB2000-INT system presented in the paper is an internet information system

designed for the retrieval of real estate cadastre data and is complementary to the main system in which cadastre database is maintained. The system has been deployed in about 50 intranets and extranets in local governments throughout Poland while the main EGB2000 cadastre system is used by above 100 centers.

The EGB2000-INT system has been implemented using PHP script language and accommodated for cooperation with Apache or IIS web servers. It assures communication with Ms SQL Server and MySQL database management systems. Using the system lies mainly in formulating queries, browsing the list of retrieved objects, choosing the objects to reports and generating reports in pdf format

At present data in cadastre systems are not complete yet. However descriptive data of land premises are fully complete, but at present information centers are gathering the data of buildings and apartments and prices of premises. Numeric plans of real estate are being created or complemented too. The generalized structure of pages in the EGB2000-INT system is shown in Figure 1, where M denotes the page of main menu, numbered ovals represent the pages with search criteria adequate for individual functions of the system and O indicates pages with retrieved objects.

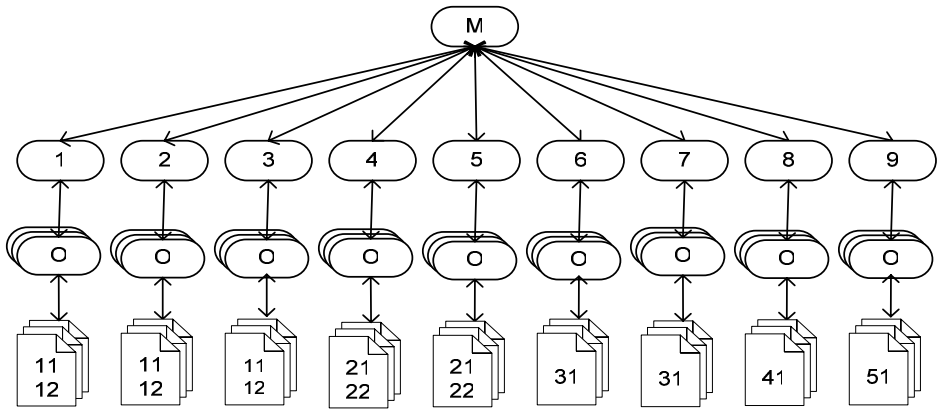


Figure 1. Reports implemented in the EGB2000-INT system

The names of system functions and the explanations of report codes are listed in Table 1.

Table 1. List of system functions and reports implemented

Option	Name of the option	Reports
1	Land registration unit search	11, 12 – extracts from land register
2	Parcel search	11, 12 – extracts from land register
3	Parcel search based on a list	11, 12 – extracts from land register
4	Building registration unit search	21, 22 – extracts from building register
5	Building search	21, 22 – extracts from building register
6	Apartment registration unit search	31 – extracts from apartment register

7	Apartment search	31 – extracts from apartment register
8	Price and value of premises search	41 – extracts from price register
9	System usage monitoring	51 – usage monitor

The access to the system is limited. Each user should be registered in the system and be assigned the rights to the data from a given territory. The users of the system are the workers of local governments who utilize data to prepare administrative decisions, to inform real estate owners and to prepare reports for management boards of local governments.

2. Investigation of the EGB2000-INT System Usage

In order to discover how heavily reports are used web logs of the EGB2000-INT system gathered during 2004 have been investigated. Usage statistics of the most frequently used reports in three information centers in Poland during 12 months of 2004 are presented in the left graph in Figure 2. These reports are an extract from land register (11) and a simplified extract from land register (12). The right graph in Figure 2 shows monthly usage of all reports. It is expected that the usage of the system will increase significantly as the number of buildings and apartments registered grows.

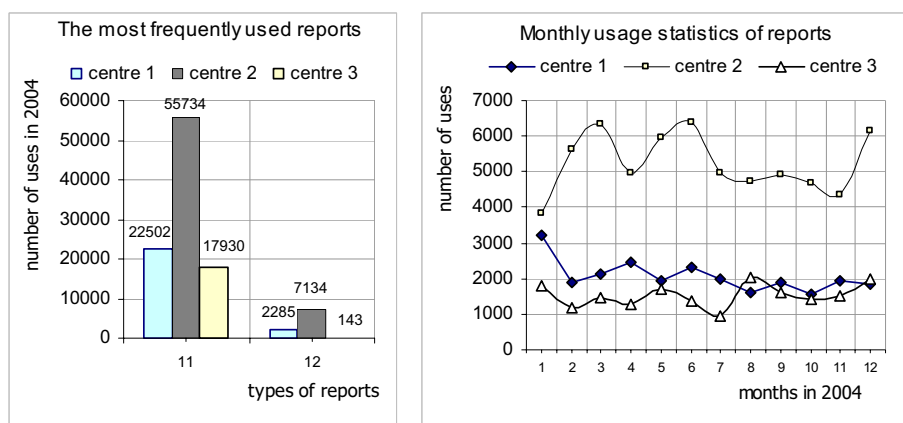


Figure 2. The most frequently used reports and monthly usage statistics of reports in 2004

The server logs were also analyzed to reveal the heaviest hourly usage of reports in the system. The top five results for each centre are shown in Figure 3. Data presented in the left graph in Figure 3 indicate that the maximum hourly usage reached 122 what equals to only 2 per one minute. In turn, in the right graph can be noticed that the biggest number of simultaneously working users was 27 during one hour in 2004.

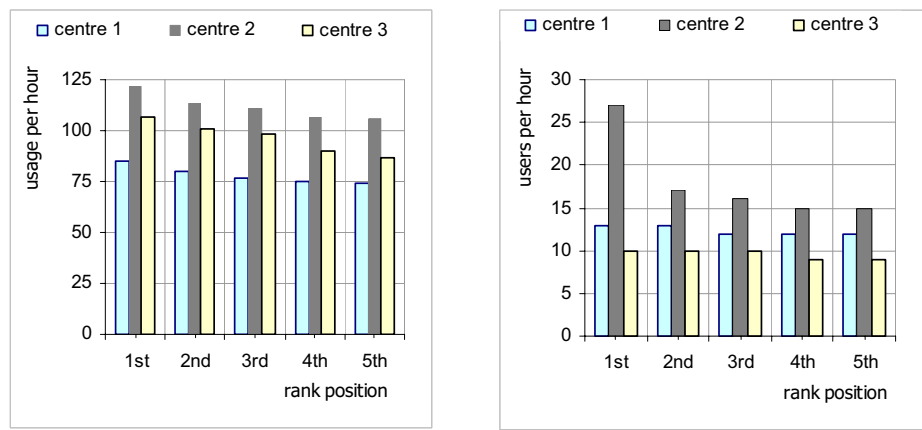


Figure 3. The heaviest hourly usage of the cadastre system

3. Overview of Reporting Tools Used in Information Systems

On the market there are many reporting tools. Selected features of some of them are presented in Table 2. They can take data from various sources such as ODBC, JDBC, EJB, OLE DB, Oracle, DB2, Ms SQL Server, Ms Access, BDE, XML, txt and others. Almost all of them allow to export reports to such file formats as html, pdf, xml, rtf, xls, csv, txt and some to doc, LaTeX, tiff, jpeg, gif. Prices are only illustrative and were selected from massive lists of prices. However the price had decisive influence on the selection of reporting tools for our internet system. Commercial tools turned out to be too expensive for local governments and it was the main reason for that we have designed and programmed reporting mechanisms based on open source tools.

Table 2. Selected features of reporting tools

Tool name	Producer	Operating system	Programming environment	Price
Active Reports .NET	Data Dynamics	Windows	MS Visual Studio.NET	\$9099
Crystal Reports 10	Seagate Software	Windows	any	\$16910 (10 users)
DataVision	Jim Menard	Windows, Linux	Java	open source
FastReport	Fast Report	Windows	Borland Delphi	\$3490
Formula One e.Report	Actuate	Windows	Java	\$2495 (25 users)
IntelliView Suite	IntelliView	Windows, Linux	MS Visual Studio.NET, Java, COM	free
JasperReports	Teodor Danciu	Windows, Linux	Java, MS Visual Studio.NET	open source

JReport	Jinfony	Windows, Linux	Java	\$19000 (server)
Rave	Nevrona	Windows	Borland Delphi, C++ Builder	\$1490
Report Builder Pro	Digital Metaphors	Windows	Borland Delphi	\$990 (server)
Report Sharp-Shooter 1.8	9Rays.NET	Windows	MS Visual Studio.NET	\$4990
ReportNet	Cognos	Windows, Linux, UNIX	any	\$4500 - \$10000
SQL Server 2000 Reporting Services	Microsoft	Windows	MS Visual Studio.NET	within SQL Server 2000 lic.
StyleReport	InetSoft	Windows, Linux	Java	\$4995 (server)

The selection process should also take into account results of benchmark tests carried out on reporting tools by their producers or by independent organizations [2], [3], [5], and [6]. Some of the results published are presented in Figure 4 for Actuate iServer, in Figure 5 for Cognos ReportNet and in Figure 6 for Crystal Enterprise 9.0. All tests have proved that commercial tools have considerably exceeded the requirements of the EGB2000-INT cadastre system.

Two series of tests performed by Actuate [3] using a 500 page report that contained an average of 101 controls, plus one static image per page are reported below. The Viewing test simulated multiple users simultaneously logging in and conducting different types of viewing requests on randomly selected pages of pre-generated reports and the On-Demand Reporting test simulated multiple users logging in, where each generated a report on demand and then viewed the report. The results shown in Figure 4 revealed that up to 9000 users can be simultaneously logged in and viewing requests while meeting the 5-second response time criteria and up to 1800 concurrent users can execute and view reports with the 10-second response time limit. As far as the throughput measure is concerned the iServer demonstrated near-perfect linear scalability.

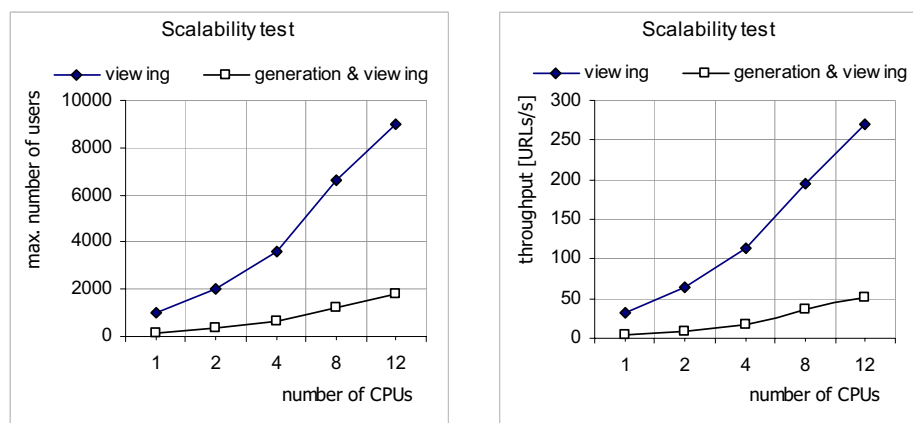


Figure 4. Results of benchmark tests on Actuate iServer (source [3], Figures 3, 4, 5, 6)

The tests accomplished by Cognos [5] were designed to simulate an enterprise reporting environment in which users navigated a Web portal to view and execute reports. For all tests Windows Server 2003 Enterprise Edition as operating system was used and Microsoft IIS 6.0 as web server, IBM WebSphere as application server, IBM DB2 as database system and Mercury Interactive Loadrunner 7.8 as a load tool. In Figure 5 the results for two configurations are presented. The first configuration with 44 processors assures average report generation times less than 25 seconds for 1100 concurrent users whereas the second with 16 CPUs achieves similar times up to 500 concurrent users.

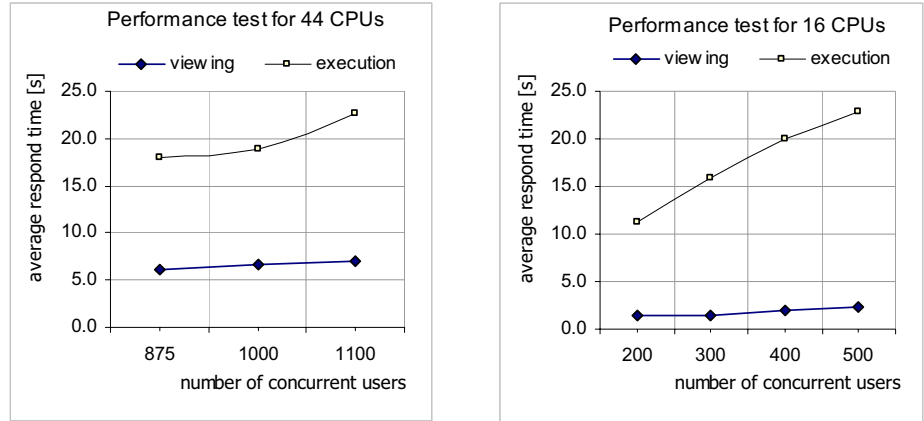


Figure 5. Results of benchmark tests on Cognos ReportNet (source [5], Tables 5, 6)

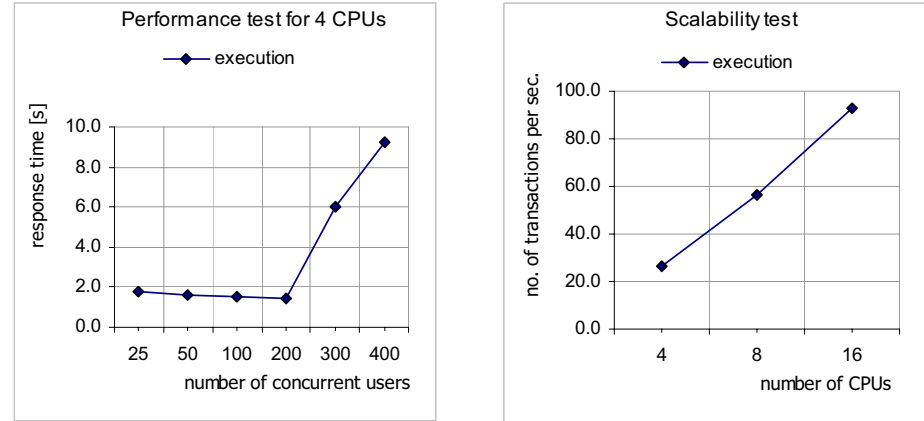


Figure 6. Results of benchmark tests on Crystal Enterprise 9.0 (source [6], Figures 2, 4)

The tests carried out by Crystal Decisions [6] used different scripts that were created with Rational Test Robot to model real world usage patterns. The scripts were

randomly chosen by virtual users. The test application run on IBM's @xSeries 16-way with Windows Server 2003. The results (see Figure 6) proved that with 4 processors response time started worsening when the number of concurrent users was bigger than 200. In turn the scalability expressed in terms of the number of transactions per second was linear.

4. Testing of Reporting Mechanisms Implemented in the EGB2000-INT System

The analysis of the prices of reporting tools and the results of benchmark tests shows that these tools are rather designed for large scale internet applications. Their performance exceeds considerably the requirements of our cadastre system and the prices could not be accepted by Polish local governments. Hence the authors of the system were forced to look for less expensive solutions. During last four years three versions of reporting mechanisms were chosen and implemented in the EGB2000-INT system, i.e. mechanisms based on Crystal Reports [16], Free PDF library [13] and XML technology [11]. Each solution was tested using Web Application Stress Tool in order to determine what limits in scalability and efficiency could be observed and finally to decide whether the system could be deployed and exploited by information centers in Poland. The most frequently used report i.e. an extract from land register unit was the main subject of all tests.

The Web Application Stress Tool is a Microsoft's product which allows testing the performance of internet services due to simulating a great number of users. The tool enables to record all requests and server responses during a session in form of a script. The scenario can also be created and edited manually or using IIS log files. Having recorded the script the tester can determine test parameters including the number of iteration, the number of users (threads), connection bandwidth, and time of a test.

During testing the heaviest load of the server is simulated. The Web Application Stress loads the server continuously, which means the tool sends a request, the server processes it and sends a result back and having received the response it starts immediately to generate next request. Such operation does not occur during normal work of the system. More probably, when a user receives the page with retrieved data, he starts to browse the result and only after some time he formulates next query.

After a test is completed the Web Application Stress generates a report comprising besides general information as time, number of users and requests also detailed figures of e.g. how many times each page was demanded, what was the server response time, how many errors were generated by server.

4.1. Testing of Reporting Mechanism Using Crystal Reports

The architecture of reporting mechanism using Crystal Enterprise 8.5 implemented in the EGB2000-INT system is shown in Figure 7. The configuration of the experiment was as follows. Intel Pentium 600 MHz and 256 MB RAM was used as the server. Windows 2000 Server with IIS web server were installed at the same computer as Oracle 8.1.6 and Ms SQL Server 2000. Both databases contained the same data [16].

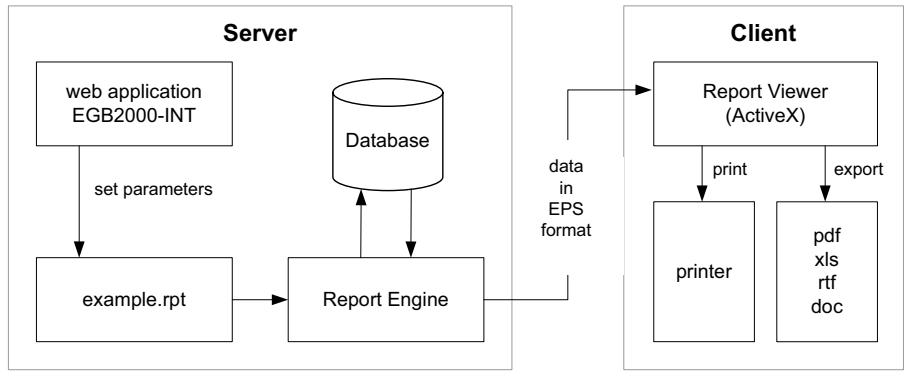


Figure 7. Architecture of reporting mechanism using Crystal Reports

In order to be closer to the real activity of users the test scenario comprised calls for three reports using different system options. The tests simulated continuous work of 1, 3, 6 or 12 users and lasted 1 minute. Each test was repeated three times and the mean value of each result was calculated. The results presented in Figure 8 allowed us to state that reporting solutions were efficient enough to cope with assumed system load in both cases of Oracle and SQL Server databases [16].

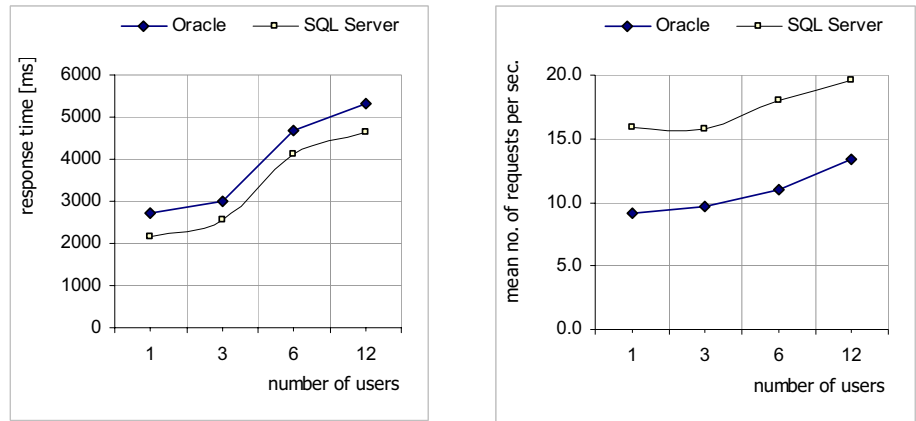


Figure 8. Test results of reporting mechanism using Crystal Reports

4.2. Testing of Reporting Mechanism Using Free PDF Library

In Figure 9 the architecture of reporting mechanism using Free PDF library implemented in the EGB2000-INT system is presented. The server used was Intel Pentium 1433 MHz and 512 MB RAM. Four configurations of operating system, application technology, web server and DBMS were tested: (1) Windows 2000 Server, PHP, Apache and SQL Server 2000, (2) Windows 2000 Server, PHP, Apache and MySQL,

(3) Linux, PHP, Apache and MySQL – an open source solution, (4) Windows 2000 Server, ASP, IIS, SQL Server 2000 and Crystal Reports as reporting tool. In each case database contained the same data [13].

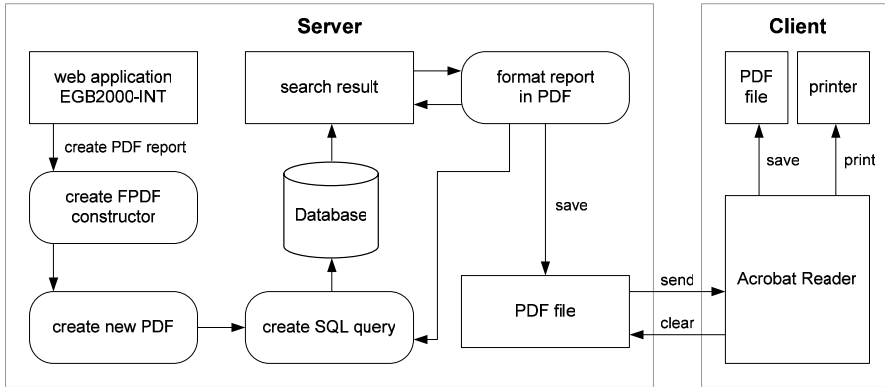


Figure 9. Architecture of reporting mechanism using Free PDF library

The test simulated continuous work of 1, 5, 10, 15 or 30 users and the test scenario comprised several steps of data retrieval and generation of reports using 12 different system options. The tests lasted 3 minutes for 1, 2, 5, 10 and 15 threads, and 5 minutes for 30 threads. The results presented in Figure 10 indicated that an open source reporting solution (3) managed to serve the system load easily while the configuration with Crystal Report (4) did not cope with 15 and more users [13].

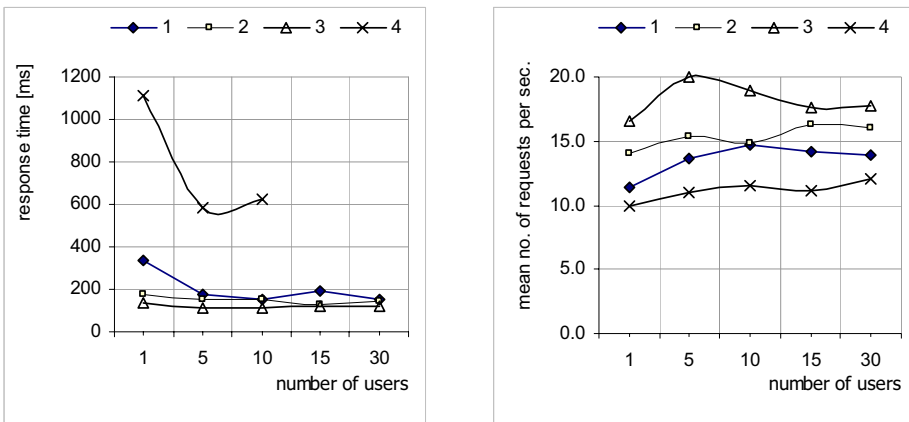


Figure 10. Comparative tests of reporting mechanism using Free PDF library

4.3. Testing of Reporting Mechanism Using XML Technology

In Figure 11 the architecture of reporting mechanism using XML technology implemented in the EGB2000-INT system is presented. The following hardware was used

in the experiment: Intel Pentium 2.8 GHz and 2 GB RAM was used as the server. Windows 2000 Professional with Apache web server were installed at the same computer as Ms SQL Server 2000 DBMS [11].

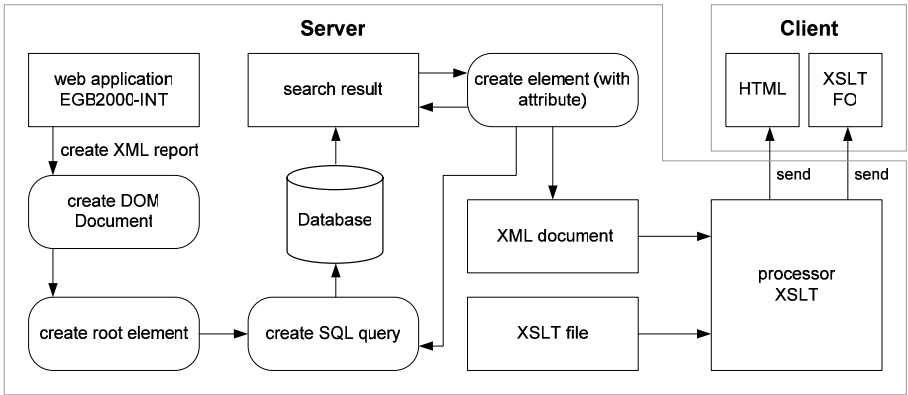


Figure 11. Architecture of reporting mechanism using XML technology

In the first series of tests two solutions based on Free PDF library and on XML technology were compared employing Microsoft’s Web Application Stress Tool. All tests were carried out with the same database and using the same scenario. Continuous work of 1, 5, 10, 15 or 30 users was simulated to reflect small, medium and heavier load of the system. Each test lasted 10 minutes. The results for the report of an excerpt from land register comprising 10 parcels are presented in Figure 12. They showed that both web applications operated similarly. However when the number of users exceeded 15 the tests produced strange results leading to the conclusion that the web server did not manage to process the load [11].

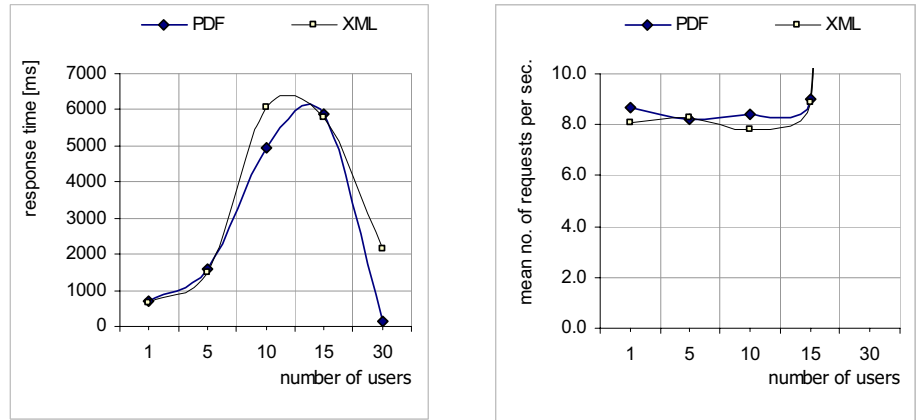


Figure 12. Comparative tests of reporting mechanism using PDF and XML technology

In the second series of tests reports retrieving 1, 10, 100 and 500 parcels were compared. In this experiment growing load of the system from normal to excessive was

simulated. As it can be seen in Figure 13 system malfunctions when the number of users is bigger than 15. The capabilities of the system seemed to be overridden when the number of records was equal to 500 and the number of users was greater than 5 [11]. Sometimes when the number of users was bigger than 15 the Windows operating system could not remove all threads from memory and therefore the results of the tests were undetermined.

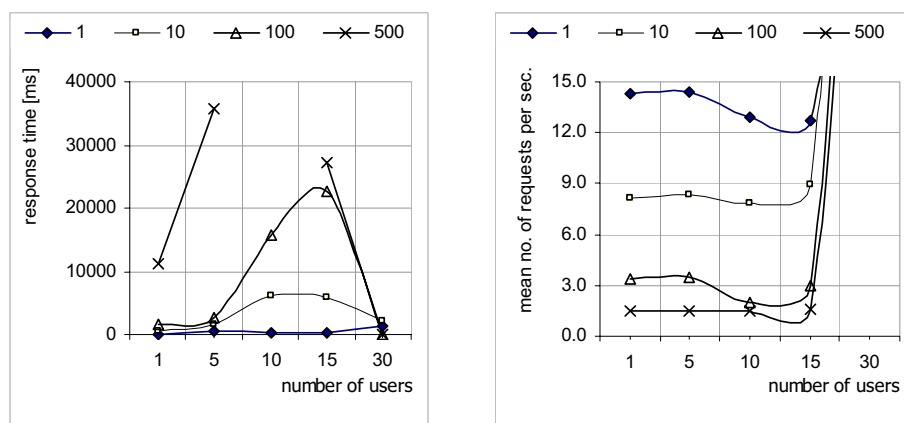


Figure 13. Test results of reporting mechanism using XML technology

5. Conclusions and Future Works

The tests reported above have proven that all three reporting mechanisms fulfill the system requirements as far as efficiency and scalability are concerned. Then the cost factor had the decisive influence on applying open source tools for the cadastre information system. Local governments have forced the implementation of the least expensive solution using PHP technology in Linux environment. Two first solutions have been exploited at local governments for three years and they have proven their usefulness and sufficient performance. The tests have revealed the limits of XML based reporting tools. Nevertheless those limits are at present far beyond the real usage of the system. In the next future XML technology will be developed in order to improve system performance as well as to implement maps into reports in the system.

References

- [1] A Guide to the Project Management Body of Knowledge (PMBOK Guide). Project Management Institute (2000)
- [2] Actuate 7 iServer vs. Crystal Enterprise 9. An Interactive Reporting Performance Benchmark Project Final Report. *National Technical Systems* (2003)
- [3] Actuate 7SP1 Performance and Scalability on Windows 2003. *Actuate Corporation* (2004)
- [4] Cadle J., Yeates D: Project Management for Information Systems. Pearson Education Limited (2001)
- [5] Cognos ReportNet Scalability Benchmarks – Microsoft Windows. *Cognos* (2004)

- [6] Crystal Enterprise 9.0 Baseline Benchmark: Windows Server 2003 on IBM@server xSeries 16-way. Crystal Decisions (2003)
- [7] Górski J. (red.): Inżynieria oprogramowania w projekcie informatycznym (in Polish). *MIKOM* (2000)
- [8] Leffingwell D., Widrig D.: Managing Software Requirements. A Unified Approach. *Addison-Wesley* (2000)
- [9] Leishman T.: Extreme Methodologies for an Extreme World. *CROSSTALK The Journal of Defense Software Engineering* Vol. 14 No. 6 (2001)
- [10] Linscomb D.: Requirements Engineering Maturity in the CMMI. *CROSSTALK The Journal of Defense Software Engineering* Vol. 16 No. 12 (2003)
- [11] Podyma M.: Review and comparative analysis of reporting tools for internet information systems. M.Sc. Thesis (in Polish). Wrocław University of Technology (2005)
- [12] Pressman R. S.: Software Engineering: A Practitioner's Approach. *McGraw-Hill* (2001)
- [13] Skupień, M.: An Internet Information System of Parcels and Buildings in Linux Environment. M.Sc. Thesis (in Polish). Wrocław University of Technology (2003)
- [14] Sommerville I.: Software Engineering. *Addison-Wesley* (2004)
- [15] SWEBOK. Guide to the Software Engineering Body of Knowledge. 2004 Version. *IEEE Computer Society* (2004)
- [16] Włochowicz S.: An Internet Information System of Parcels, Buildings and Apartments using Oracle and Ms SQL Server Databases. M.Sc. Thesis (in Polish). Wrocław University of Technology (2002)

7. Software Modeling and Verification

This page intentionally left blank

Use-Cases Engineering with UC Workbench

Jerzy NAWROCKI and Łukasz OLEK

Poznań University of Technology, Institute of Computing Science

ul. Piotrowo 3A, 60-965 Poznań, Poland

e-mails: {Jerzy.Nawrocki, Lukasz.Olek}@cs.put.poznan.pl

Abstract. Use-case methodology is widely used for writing requirements. Unfortunately, there are almost no tools supporting the usage of the use cases in any software projects. UC Workbench is a tool that tries to fill in this gap. It contains a use-case editor (it spares up to 25% of the effort at introducing new use-cases and up to 40% at updating them), a generator of mockups (a mockup generated by UC Workbench animates the use-cases and illustrates them with screen designs) and Software Requirement Specification documents, and an effort calculator, based on Use-Case Points.

Introduction

Use cases have been invented by Ivar Jacobson [10], as a way of specifying functional requirements. Later on they have been incorporated into the Unified Software Development Process [11] and the Rational Unified Process [13]. Since then, they have been gaining more and more popularity. A significant step forward, in the development of the use-case methodology, has been taken by Alistair Cockburn and his colleagues, who proposed the set of the use-case patterns [1], [4]. Unfortunately, the use-case methodology is not adequately supported by the existing tools. Even editing of the use-cases is a problem. The most popular form of the use cases is a sequence of steps specifying the so-called main scenario with a number of exceptions (called also extensions) describing alternative behavior (see e.g. [1], [4], [8]). Since steps are numbered and those numbers are referenced in exceptions, deleting or inserting a step requires the re-numbering of all the subsequent steps, and accordingly all the exceptions associated with them. Unfortunately, it is not easy to find an editor that would support deleting and inserting a step into the use case. Even Rational Requisite Pro [19], a requirements engineering tool, developed by a company for which Ivar Jacobson worked for many years, does not have such functionality. According to our findings the only tool that supports the use cases is the use-case editor, offered by Serlio Software [26]. However, editing is just one, in a wide range of the use-case-related activities that could be supported by a computer. The aim of the paper is to present a tool, UC Workbench, supporting various activities concerning the use cases. UC Workbench is based on a semi-formal language, FUSE, in which the use-cases are described (see Section 1). The first and very restricted version of the tool has been described in [16]. The version of UC Workbench presented in this paper supports the following use case engineering activities:

- Editing of use cases (see Section 2).
- Performing automatic reviews (also Section 2).
- Generating a mockup (Section 3).
- Composing a software requirements specification document (SRS document for short) compliant with IEEE Std 830 [9] (Section 4).
- Generating effort calculators based on Use Case Points [11], [18] that support defining the scope of a project stage (Section 5).

What is important, in UC Workbench, is the fact that there is only one representation of a set of use cases describing a given system, and from that representation a mockup, an SRS document, and effort calculators are derived. It further assures that all those artifacts are always consistent one with the other. In Section 6 a simple quasi-experiment is described, which aimed at checking if UC Editor (which is part of UC Workbench) can be really helpful.

1. FUSE: A Language for Use-Cases Description

The use cases collected by the analyst are edited with the help of UC Editor, a part of UC Workbench. The editor uses FUSE (Formal USE cases) language. It is a simple language formalizing structure of use-cases description to allow generating of mockups and effort calculators (actor descriptions and steps within the use cases are expressed in a natural language).

FUSE is based on use-case patterns collected by Adolph and his colleagues [1]. Use-case written in FUSE follows ScenarioPlusFragments pattern: the description consists of the main scenario split into a number of steps and extensions expressed in the natural language. To make the formal description of the use-cases more precise and readable we have decided to introduce the Either-Or and repeatable steps constructs to FUSE. Moreover, for the sake of readability, FUSE allows nested steps that are especially helpful when combined with the above constructs.

One can argue that the above constructs can be difficult for some end users to understand. In the case of UC Workbench it should not cause problems, as the use cases are accompanied by an automatically generated mockup, which visualizes the control flow by animating the use cases.

1.1. Repeatable Steps

Sometimes a step can be repeated more than once in a scenario. Assume that using the use-case *UC-2: Adding a book to the cart* someone wants to describe how to buy books in an Internet-based bookstore. Writing this in the following way:

1. Customer adds a book to the cart (UC-2).
2. Customer pays for the selected books.

is not fully correct, because it suggests that the Customer can add only one book to the cart. We can try to fix it by adding the extension:

- 1a. Customer wants to add another book.
 - 1a1. Return to step 1.

Unfortunately, that looks quite artificial and would clutter the description. To solve the problem we have decided to mark steps that can be repeated more than once with a star (*) sign. Now, we can describe the situation in the following way:

1. * Customer adds a book to the cart (UC-2).
2. Customer pays for the selected books.

1.2. Either-Or

This construct was introduced to express a nondeterministic choice between alternative steps (more details and motivation can be found in [16]). The situation of buying books in a bookstore (extended with the removing a book from the cart) can be described in the following way:

1. Either: Customer adds a book to the cart (UC-2).
2. Or: Customer removes a book from the cart.
3. Or: Customer pays for the books in the cart.

1.3. Nested Steps

Sometimes a single step can be subdivided into 2 or 3 other steps. Then it can be convenient to have the “substeps” shown directly in the upper level use case, without the necessity of creating the use case on lower level. For instance, assuming that *UC-2: Adding a book to the cart* it has only two steps buying books could be described in FUSE in the following way (use-case header and extensions have been omitted):

1. Either: Customer adds a book to the cart.
 - 1.1. Customer selects a book.
 - 1.2. System shows new value of the cart.
2. Or: Customer removes a book from the cart.
3. Or: Customer pays for the books in the cart.

2. UC-Editor and Automated Reviews

Figure 1 presents a UC-Editor’s screen with two exemplary uses of the cases written in FUSE: *Running an Online Bookshop* and *Ordering Books*. Each use case is assigned a tag (B1 or U1). While editing use cases, step numbers are not shown on the screen (but they appear in the specification document and in the mockup). References between steps are described by means of labels.

In an ideal case UC-Editor would not only support the editing of the use cases but it would also check their quality automatically. Obviously, that is not possible. However, the FUSE language presented in the previous section allows automatic detection of the following “bad smells” concerning use cases:

	A	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	
1	UC#	Label	Title	Step																			Precondition
2	B1		Running an Online Bookshop																				
3				Bookshop gathers information about new books from publishers and place it online for all customers																			
4				Customer visits the online bookshop and orders books he wants to. He chooses best delivery method and																			
5				Bookshop gathers the books: either from the storehouse or directly from the publishers																			
6				Bookshop packs all the books, includes the invoice for customer and sends using choosen delivery method																			
7	U1		Ordering books																				Customer is
8				Customer opens main page of the Bookshop																			
9				System presents a list of categories and all new and top positions																			
10		Adding		* Customer is composing his order																			
11				Either: Customer adds a book to his cart																			
12				Customer chooses desired book																			
13				System shows book details																			
14				Customer adds the book to cart																			
15				Or: Customer removes one book from cart																			
16				Customer finalizes the order																			
17				Exception: The cart is empty																			
18				System shows appropriate message and goes back to [step.Adding]																			

Figure 1. UC-Editor screen with two use cases

- *Stranger* – An actor appearing in a step has not been defined (he is not on the actors list). Sometimes it is caused by using two different words for the same role (e.g. *customer* and *user*).
- *Lazy actor* – An actor is on the actor's list but he does not appear in any step.
- *Too short or too long scenarios* – A scenario contains less than 3 or more than 9 steps [1].
- *Too many extensions* – The total number of steps within the main scenario and in all its extensions is greater than a threshold (in our opinion it should be 16).
- *Incomplete extensions* – An extension contains an event but no steps associated with it.
- *Too much happiness* – More than 50% of the use cases have no extensions (the description is dominated by “happy-day scenarios”).
- *Dangling functionality* – A given system-level use case is not used by any of the business-level use cases.
- *Outsider* – A given business-level use case is not supported by any system-level one (it appears to be outside of the system scope, at least for the time being).

The above mentioned bad-smell detection is intended to support the two-tier-review approach [1] to use-case quality assurance.

3. Generating of Mockups

There are two kinds of prototypes [18]: throwaway prototypes (mockups) and evolutionary ones. The latter are the core of every agile methodology. The former could be used to support customer-developers communication concerning requirements, however, their development had to be very cheap and very fast.

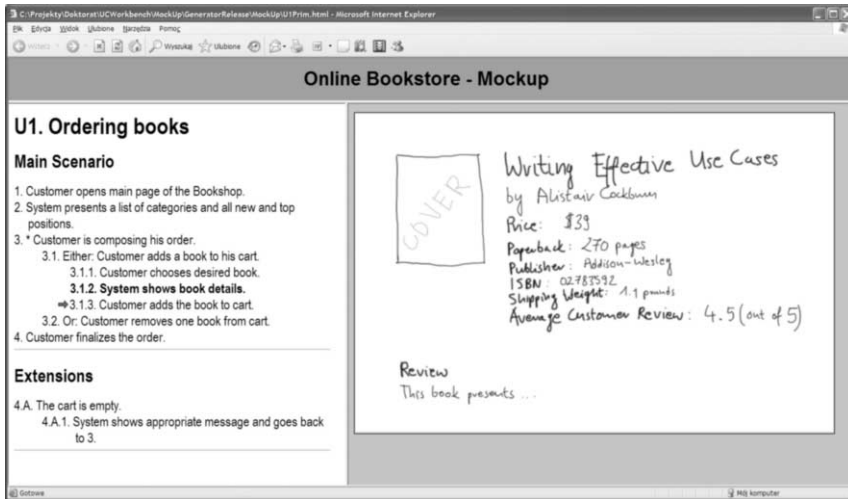


Figure 2. Example mockup

The mockups generated by UC Workbench are simple and effective. They focus on the presentation of functionality. They combine the use cases (i.e. behavioural description) with screen designs that are associated with them (that complies with the Adornments pattern [1]). A generated mockup is based on a web browser and it consists of two windows (see Figure 2):

- the *scenario window* presents the currently animated use cases (it is the left window in Figure 2) and the current step is shown in bold;
- the *screen window* shows the screen design associated with the current step (it is the right window in Figure 2).

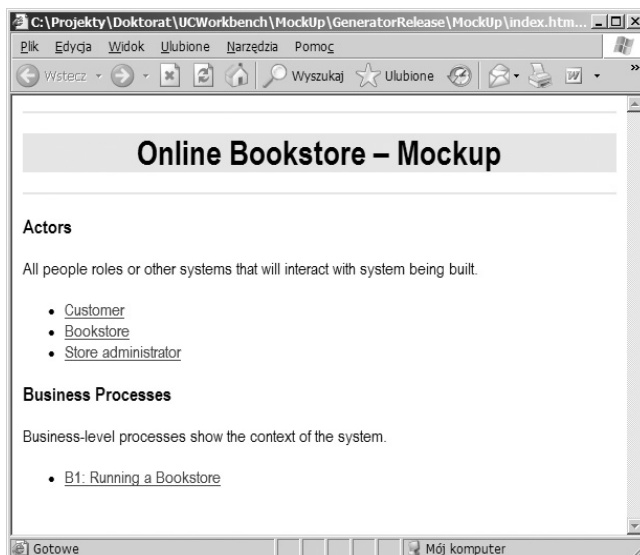


Figure 3. Choosing actor or business process

The animation process starts with the presentation of all the actors and all the business-level use cases (see Figure 3). By clicking an actor one can get the actor description and the list of the use cases in which the actor participates in (see Figure 4). By selecting a business-level use case one will be able to go down through the use cases DAG (direct acyclic graph) created by the *include* relation.

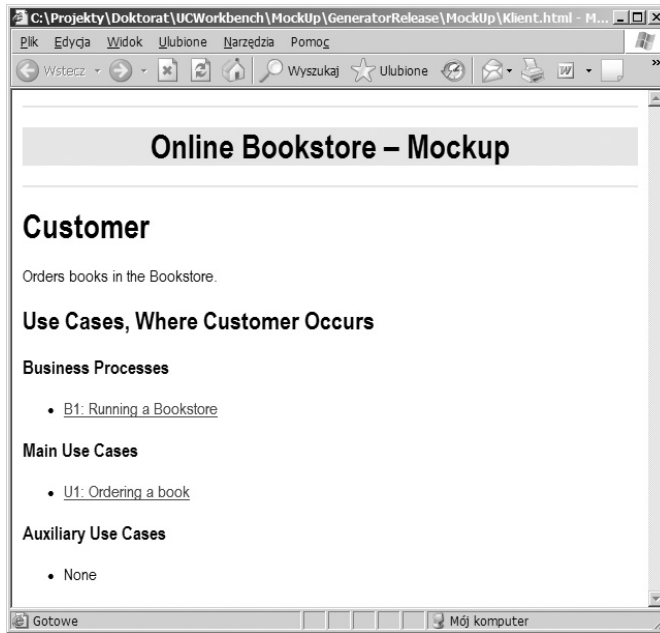


Figure 4. Actor details with the list of use cases in which the actor participates

To generate a mockup the analyst has to associate with use-case steps names of files containing screen designs. The fidelity levels of screen designs are up to the analyst (an interesting discussion about fidelity levels can be found in [21], [25], [14], [17], [23]). The screen design shown in Figure 2 is at the low fidelity level and it has been created with a tablet connected to PC. After decorating 'difficult' use-cases with screen designs one can generate a mockup "at the press of a button". Using UC Workbench one can produce a mockup (i.e. re-write use cases and adorn them with screen designs) within a few hours – that can really support customer-developers communication, especially if there is a danger of unstable, ambiguous or contradictory requirements.

4. Composing the SRS Document

Someone can think that when a mockup is available no Software Requirements Specification (SRS) document is needed (the functionality is already shown by the mockup). There are two arguments that motivated us to support the composition of the SRS documents:

- *Nonfunctional requirements.* Assume one of the requirements specifies that the system must be able to store up to 20 000 personal records. To which use case this 'adornment' should be assigned?
- *Customer's attachment for documentation.* If one wants to have a collaborating customer, he must show appreciation for his experience and believes. One of Covey's principles [7] is '*First seek to understand, then to be understood*'.

To solve those problems we have decided to extend UC Workbench with an SRS Composer. Our SRS template is based on IEEE Std 830 [9] and it consists of the following sections:

1. Introduction
2. Business Model
3. Functional Requirements
4. Nonfunctional Requirements

Sections 1 and 2 are generated directly from the use-case descriptions. The Business Model contains subsections describing actors, business-level use cases and business objects. The Functional Requirements contain system-level and auxiliary use-cases assigned to subsequent actors. Introduction and Section 3 are written by the analyst as separate LATEX files. Actors, business objects and the use-cases are referenced by non-functional requirements via tags. If a non-functional requirement references e.g. a use-case that has been removed then the analyst will get a warning.

5. Effort Estimation and Scope Planning

UC Workbench supports effort estimation and scope planning based on use cases. As a framework we have chosen Use-Case Points proposed by Gustav Karner [12], [20]. In our approach scope planning comprises three layers: Planning Game borrowed from XP (highest layer), Wideband Delphi, and automatic effort calculation (lowest layer). Automatic effort calculation (AutEC) provides initial values that are later modified by experts during Wideband Delphi session. Such a session is part of a Planning Game which provides the customer (or his representative) with information necessary to define the scope for the next development stage.

XP's Planning Game is played on a table on which story cards are put. In our approach user stories are replaced with system-level use cases and the table is replaced by a computer running AutEC (AutEC is a specialized spreadsheet which relieves estimation experts from tedious calculations).

The original Use-Case Points help to estimate effort for a system defined by a set of use cases. We have adjusted the method to allow to estimate effort for single use cases and then to define scope of the increment by selecting most important – from customer's point of view – use cases (their total effort cannot exceed man-hours allocated for the increment).

The main part of AutEC screen consists of rows corresponding to the use-cases. Each row contains:

- *use-case name* with a hyperlink to use-case text,

- *effort estimated by AutEC* used by experts as a starting point for their discussion (see below),
- *effort estimated by experts* and following from their discussion,
- *summary of experts estimations* presenting the most optimistic (O), most pessimistic (P), and average effort (A) accompanied with most probable one equal to $(O + 4A + P)/6$.

AutEC can estimate the effort only for use-cases in the formal form (see Section 1). Use-case effort is computed by AutEC using the following formula derived from Use-Case Points:

$$UseCaseEffort = 25 \cdot UCW(s) \cdot TF \cdot EF$$

UCW is Use-Case Weight and it depends on the number of steps, *s*, in a given use-case. Originally *UCW* was equal to 5, 10 or 15 points depending on the number of “transactions” (here use-case steps). For up to 3 steps *UCW* was equal to 5, and for 8 or more steps *UCW* was 15. It means that reducing a 7-steps-long use-case by 2 steps has no meaning but reducing a 4-steps-long use-case just by 1 step can decrease its complexity by half. To remove this effect, we have changed this rule and defined *UCW* in the following way:

$$UCW(s) = 17 - 20 \cdot \exp^{-s/5}$$

where *s* represents number of steps. For the proposed function the average value of *UCW* for 2 and 3 steps is almost 5 (4.81), and for 4, 5, 6, and 7 steps it is about 10 (10.2) — the same holds for the original *UCW* function proposed by Karner.

TF and *EF* denote technical and environmental factors (*TF* depends on 13 factors, and *EF* on 8). The factors are evaluated by the experts and are the subject of their discussion. All the changes made to *TF* and *EF* factors managed by AutEC are immediately visible on the use-case rows in the main part of the AutEC screen. This allows the experts to speculate about influence of non-functional requirements on the effort.

The original Use-Case Points depend not only on use-cases but also on the complexity of the actors. There are three kinds of actors: simple (API interface), average (TCP/IP or text interface), and complex (GUI interface). AutEC assigns effort to actors in the following way:

$$ActorEffort = 25 \cdot AW \cdot TF \cdot EF$$

where *AW* equals 1 for simple actors, 2 for average, and 3 for complex. AutEC shows the effort connected with actors in a separate row on the main AutEC screen. Only actors involved in a selected set of use-cases are taken into account.

6. Evaluation of the Tool

In December 2004 we conducted an experiment comparing the effort needed to prepare the use cases using a general-purpose text processor, MS Word, and *UC Editor*.

Twelve students participated in the experiment. We have divided them into two groups: six people were using MS Word and remaining six *UC Editor*. Students were provided with drafts of 4 use cases (each of them contained 6-9 steps), and their task was to Prepare the use-case-based specification using the assigned tool. The task took from 40 to 80 minutes. At the next stage the students had to introduce some changes to the specification.

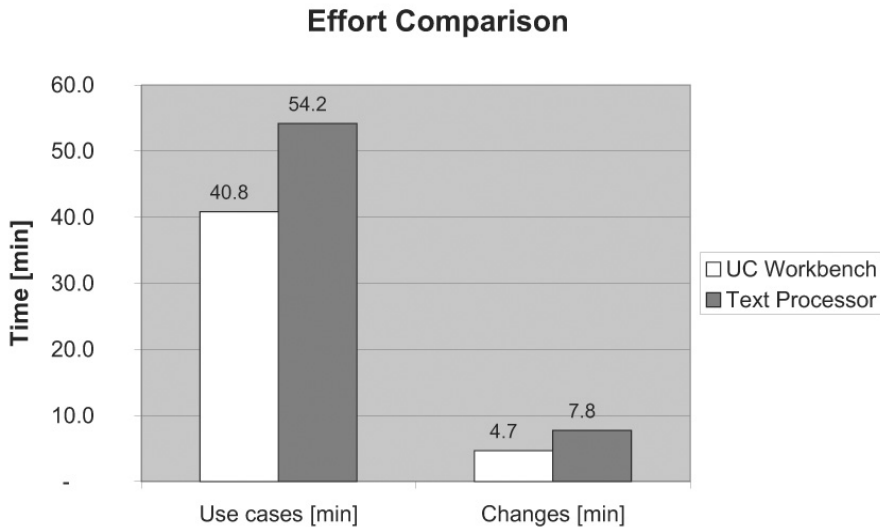


Figure 5. Average effort necessary to prepare use cases using MS Word and UC Editor

It turned out (see Figure 5) that using UC Editor instead of MS Word one can save, on average, 25% of time (the standard deviation for *UC Editor* was 4.2 and for MS Word it was 17.2). The relative savings are even greater when one has to maintain use cases. Due to our experiment, one can save 40% of time (on average) by using UC Editor instead of a general purpose editor (the standard deviation for *UC Editor* was 1.6 and for MS Word it was 1.5).

We must emphasize that we measured only the effort of writing use cases, so it does not mean that the whole phase of requirements elicitation will take 25% less, when using UC Editor.

7. Conclusions

UC Workbench presented in the paper supports the editing the use cases (we were surprised that Rational Requisite Pro much more supports “traditional” requirements than use cases), generates mockups that animate use cases, and creates effort estimators adjusted to the current set of use cases. Moreover, UC Workbench can generate Requirements Specification Document complying to one of the templates recommended by IEEE Std 830 [9]. What is important for agile developers, all those artifacts (i.e.

mockups, estimators, and requirements documents) can be obtained automatically and each of them is consistent with the others. Early experience, based on university experiments and observations of projects performed both at academia and in industry setting, is quite promising. UC Workbench can save about 25% time when writing use cases from scratch and about 40% when modifying existing ones.

In the nearest future we are going to extend effort estimators (AutEC) with a historical database showing for past projects the estimated and actual effort, technology used in the project, and people involved in it. That should help the experts to provide better effort estimates.

Acknowledgements

We would like to thank our students, who participated in the experiment. We also thank Grzegorz Leopold and Piotr Godek from PB Polsoft – their brevity allowed us to get a feedback from industry and helped us to improve the tool.

This work has been financially supported by the State Committee for Scientific Research as a research grant 4 T11F 001 23 (years 2002-2005).

References

- [1] Adolph S., Bramble P., Cockburn A., Pols A.: Patterns for Effective Use Cases. *Addison-Wesley* (2002)
- [2] Beck, K.: Extreme Programming Explained. Embrace Change. *Addison-Wesley*, Boston, 2000
- [3] Boehm, B., Turner, R: Balancing Agility and Discipline. A Guide for the Perplexed. *Addison-Wesley*, Boston, 2004
- [4] Cockburn A.: Writing Effective Use Cases *Addison-Wesley*, *Addison-Wesley*, Boston, (2000)
- [5] Cockburn, A.: Agile Software Development. *Addison-Wesley*, Boston, 2002.
- [6] Cohn, M.: User Stories Applied. *Addison-Wesley*, Boston, 2004.
- [7] Covey, S.: The Seven Habits of Highly Effective People. *Simon and Schuster*, London, 1992.
- [8] Fowler, M., Scott, K.: UML Distilled. *Addison-Wesley*, Boston, 2000.
- [9] IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications. *The Institute of Electrical and Electronics Engineers, Inc.* (1998)
- [10] Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach. *Addison-Wesley*, Reading MA, 1992.
- [11] Jacobson I., Booch G., Rumbaugh J.: The Unified Software Development Process. *Addison-Wesley*, Reading, 1999.
- [12] Karner G.: Use Case Points - Resource Estimation for Objectory Projects. *Objective Systems SF AB*, 1993.
- [13] Kroll, P., Kruchten, Ph.: The Rational Unified Process Made Easy. *Addison-Wesley*, Boston, 2003
- [14] Landay J.A.: SILK: Sketching Interfaces Like Crazy IEEE Computer-Human Interaction (April 13-18, 1996)
- [15] Larman, C.: Agile And Iterative Development. A Manager's Guide. *Addison-Wesley*, Boston, 2004.
- [16] Nawrocki J., Olek Ł.: UC Workbench – A Tool for Writing Use Cases and Generating Mockups. Proceedings of the 6th International Conference on eXtreme Programming and Agile Processes in Software Engineering XP 2005, LNCS, *Springer Verlag*, 2005 (in print).
- [17] Newman M.W., Landay J.A.: Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. *Proceedings of the Conference on Designing interactive systems: processes, practices, methods, and techniques* (2000), p. 263 - 274
- [18] Pressman, R.: Software Engineering. A Practitioner's Approach. *McGraw-Hill*, New York, 1997
- [19] Rational Software Corporation: Using Rational RequisitePro. 2001.
- [20] Ribu K.: Estimating Object-Oriented Software Projects with Use Cases Master of Science Thesis, University of Oslo 2001

- [21] Rittig M.: Prototyping for Tiny Fingers. *Communications of the ACM*, April 1994/Vol. 37, No. 4 p. 21-27
- [22] Schwaber, K.: Agile Project Management with Scrum. Microsoft Press, Redmond, 2004.
- [23] Snyder C.: Paper Prototyping: The Fast and Easy Way to Define and Refine User Interfaces. Morgan Kaufman Publishers (2003)
- [24] Stapleton, J.: DSDM. Business Focused Development. *Addison-Wesley*, London, 2003.
- [25] Walker M., Takayama L., Landay J.A.: High-fidelity or Low-fidelity, Paper or Computer? Choosing Attributes When Testing Web Prototypes. *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting: HFES2002*. pp. 661-665
- [26] <http://www.serliosoft.com/casecomplete/>

Conceptual Modelling and Automated Implementation of Rule-Based Systems¹

Grzegorz J. NALEPA and Antoni LIGEŻA

*Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
e-mails: gjn@agh.edu.pl, ligeza@agh.edu.pl*

Abstract. The paper presents the elements of a novel approach to joint modelling and implementation process of rule-based systems. The design is performed top-down with a three-level, hierarchical approach. It covers conceptual design, logical design and physical design. A new tool, *Attribute-Relationship Diagrams* (ARD) for logical design stage are put forward. Another logical knowledge representation tool, used at logical design stage consists of the eXtended Tree Tables (XTT). The final, physical design stage consists in mapping to PROLOG meta code. The proposed approach is supported with visual design tool MIRELLA and offers possibility of amalgamating design and implementation into a single procedure. Moreover, incremental verification is also enabled.

Introduction

Knowledge-based systems are an important class of intelligent systems originating from the field of Artificial Intelligence (AI). They can be especially useful for solving complex problems in cases where purely algorithmic or mathematical solutions are either unknown or demonstrably inefficient.

The intelligent behaviour is rule-governed. Even in the common sense people have the tendency to associate intelligence with *regular behaviour*. Such regularities defining *patterns of behaviour* may often be expressed by *rules*, which are an efficient and straightforward means of knowledge representation. In AI rules are probably the most popular choice for building knowledge-based systems, that is the so-called *rule-based expert systems* [1], [2].

Although the rule-based programming paradigm seems relatively conceptually simple [1], [2], in case of realistic systems it is a hard and tedious task to design and implement a rule-based system that works in a correct way. Problems occurs as the number of rules exceeds even relatively very low quantities. It is hard to keep the rules consistent, to cover all operation variants and to make the system work according to the desired algorithm.

Classical software engineering approaches cannot be applied directly to RBS due to fundamental architectural differences between knowledge-based systems and classic software systems. In case of RBS *knowledge engineering* approaches must be used [1].

¹ Research supported from a KBN Research Project ADDER No.: 4 T11C 035 24

Practical design of non-trivial rule-based systems requires a systematic, structured and consistent approach. Such an approach is usually referred to as a *design methodology*. The basic elements distinguishing one methodology from the other are the *internal design process structure* i.e. the way of structuring the design process and the *components* of the process [1].

This paper is devoted to presenting the elements of a new hierarchical approach to rule-based systems design and implementation. Some main features of this new approach are discussed in Section 1. In subsequent sections different stages of this methodology are presented in more detail: ARD, a conceptual modelling method, is introduced in Section 2, and XTT, a logical design method, is discussed in Section 3. Section 4 presents issues concerning practical implementation of XTT-based RBSs in PROLOG. In Section 5 MIRELLA, a case tool supporting the visual XTT design process is shortly discussed. The paper ends with concluding remarks in Section 6.

1. Hierarchical Design and Implementation Methodology

The approach proposed in this paper is based on some recent research; some initial approaches were presented in [3], [4]. The presented methodology can be seen as a hierarchical, top-down knowledge engineering process oriented towards more and more detailed specification of the system under design. An important feature consists in amalgamating *verification* with *design* so that at each stage of the design process the selected system characteristics can be checked and corrected if necessary.²

The proposed approach follows the structural methodology for design of information systems. It is simultaneously a top-down approach, which allows for incorporating hierarchical design – in fact, any component can be split into a network of more detailed components, and a network of components can be grouped together to form a more abstract component. The approach covers the following design phases:

1. *Conceptual modelling*, in which system attributes and their functional relationships are identified. In this phase ARD diagrams are used as the modelling tool.
2. *Logical design*, during which system structure is represented as tabular-tree hierarchy. In this phase XTT representation is used as the design tool.
3. *Physical design*, in which a preliminary *implementation* is carried out. Using the predefined XTT translation it is possible to automatically build a PROLOG-based prototype.

Subsequent design phases of this approach are presented in Figure 1.

One of the most important features of this proposed approach is the *separation of logical and physical design*, which also allows for a transparent, *hierarchical* design process. The hierarchical conceptual model is mapped to modular logical structure. The approach could address three main problems: visual representation, functional dependency and logical structure, and machine readable representation with automated code generation.

² This is perhaps a unique implementation of the ideas first proposed in [5].

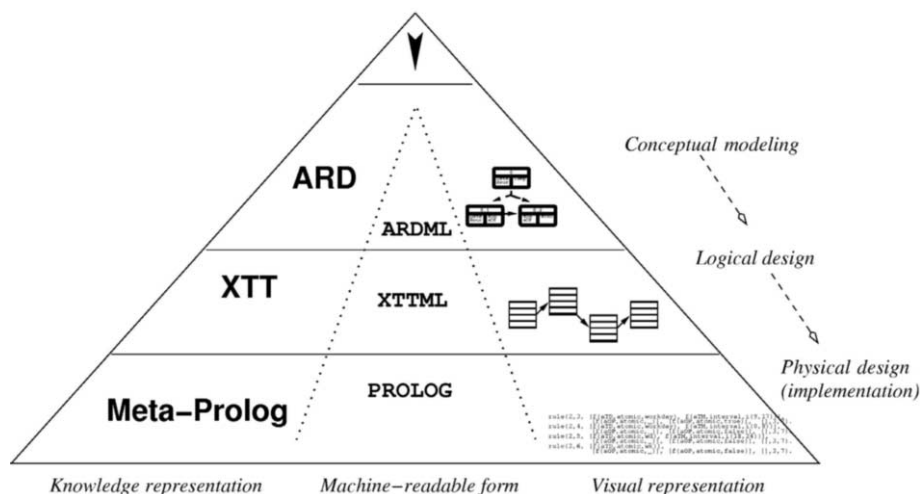


Figure 1. Phases of the hierarchical modelling, design and implementation process

2. Conceptual Modelling Using ARD

The *conceptual design* of the RBS aims at modelling the most important features of the system, i.e. attributes and functional dependencies among them. During this design phase the ARD modelling method is used. ARD stands for *Attribute-Relationship Diagrams*. It allows for specification of functional dependencies of system attributes using a visual representation.

The concept of ARD was first introduced in [4] for solving particular design problems in conceptual design of RBS. This paper provides an attempt of more general formulation of ARD as a conceptual modelling tool for RBS. ARD is incorporated as a part of an RBS design and implementation method.

The key underlying assumption in design of rule-based systems with knowledge specification in attributive logics is that, similarly as in the case of Relational Databases [6], the attributes are *functionally dependent*.

Let X, Y denote some sets of attributes, $X = \{X_1, X_2, \dots, X_n\}$ and $Y = \{Y_1, Y_2, \dots, Y_m\}$. We say that there is a functional dependency of Y from X , which is denoted as $X \rightarrow Y$, if having established the values of attributes from X all the values of attributes of Y are defined in a unique way. Further, we are interested in the so-called *full* or *complete* functional dependencies, i.e. ones such that there is $X \rightarrow Y$ but for any proper subset $X' \subset X$ it is not true that $X' \rightarrow Y$; in other words, *all* the values of attributes of X are necessary to determine the values of Y .³ A basic ARD table for specification of such a dependency is presented in Figure 2.

³ In case of a classical relational database table with the scheme specified with attributes $\{X_1, X_2, \dots, X_n, Y_1, Y_2, \dots, Y_m\}$ X will be considered as the key.

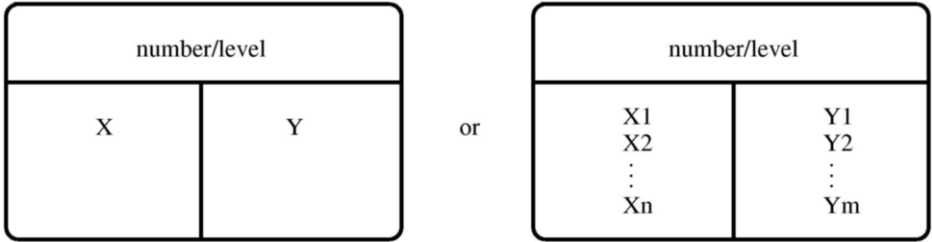


Figure 2. An ARD table: the basic scheme for $X \rightarrow Y$

In the figure, the attributes on the left (i.e. the ones of X) are the independent ones, while the ones on the right (the ones of Y) are the dependent ones. The sets of variables (e.g. X or Y) will be referred to as *conceptual variables* and they are specified with a set of detailed, atomic attributes.

An ARD *diagram* is a conceptual system model at a certain abstract level. It is composed of one or several ARD *tables*. If there are more than one ARD table, a partial order relation among the tables is represented with *arcs*. For intuition, this partial order means that attributes which are dependent in a preceding table must be established first, in order to be used as independent attributes in the following table (i.e. a partial order of determining attribute values is specified).

The ARD model is also a hierarchical model. The most abstract level 0 diagram shows the functional dependency of *input* and *output* system attributes. Lower level diagrams are less abstract, i.e. they are close to full system specification. They contain also some intermediate conceptual variables and attributes.

As it was mentioned above, system attributes can be represented as *conceptual variables* on the abstract levels. These are further specified with one or more *physical attributes* on lower level (less abstract) diagrams. In the subsequent design stages physical attributes are directly mapped into logical structure of the system and implementation. At the final, most detailed specification no conceptual variables are allowed; all the attributes must represent the physical (atomic) attributes of the system.

The table *heading* contains the table identifier which is a path-dot construction of the following recursive form:

- single number 0 is the label for the top-level diagram,
- a sequence $0.[path].m$ is the label of the m -th table at the level $length(path)+1$, where $[path]$ is empty string, a single number or a sequence composed of numbers separated with dots.

In fact, such an identifier construction defines a tree, and a particular identifier specifies a path starting from the root and identifying a table component in a unique way.

An ARD diagram of level i can be further *transformed* into a diagram of level $i+1$, which is more detailed (specific). A transformation includes *table expansion* and/or *attribute specification*. Two basic *transformations* are considered:

1. *Horizontal split*, where a table containing conceptual variables is *expanded* into two tables, containing more detailed, intermediate conceptual variables or physical attributes,

2. *Vertical split*, where a table containing two (or more) dependent attributes is expanded into two (or more) independent tables.

During the transformation a conceptual variable can be specified (substituted) by more specific conceptual variables or a physical attribute, so that in the last, most detailed level diagram, only physical attributes are present.

Roughly speaking, an ARD diagram can be considered *correct* if it is consistent with the existing functional dependencies of the attributes and the specification enables determining all the values of (output) attributes once the input values are obtained. The correctness of the final, most detailed level is translated into the following conditions:

- all of the attributes are the physical attributes of the system (no conceptual variables are allowed),
- all of the input attributes (the independent attributes in the tables to which no arcs point to) are the system inputs (they are determined outside of the system),
- any ARD table specifies a full functional dependency (local correctness requirement), and
- for any path from input to output attributes, no unestablishable attribute occurs, i.e. when traversing such a path from the left to the right, every attribute is either an input attribute or its first occurrence is as a dependent attribute in some ARD table.

The conditions above assure the possibility of determining all the values of all the attributes with respect to the functional dependencies among the attributes.

On the *machine readable level* ARD can be represented in an XML-based *ARDML* (*ARD Markup Language*) suitable for data exchange operations, as well as possibly transformations to other diagram formats.

Consider a simple but illustrative control RBS for setting the required temperature in a room, depending on the day, hour, etc. The original formulation of the example is described in detail in [7]; it was used for illustrating the XTT approach in [3], [4]. The goal of the system is to set a temperature at a certain *set point*, which is the *output* of the system. The *input* is the current time and date. The temperature is set depending on the particular part of the week, season, and working hours. An example of the ARD model can be observed in Figure 3.

The ARD hierarchy gives a conceptual model of the rule-based system. The meaning of aX is that X is a physical attribute. There are the following attributes: aSE – SEason, aOP – OPeration hours, $aTHS$ – THERmostat Setting, aTD – ToDay, and aDD – Day.

The model is based on the concepts of attributes and attribute relations. Using this model a specific *logical* structure can be designed. This structure would include actual rules and rule relations. The ARD method was invented with XTT logical design method in mind [4].

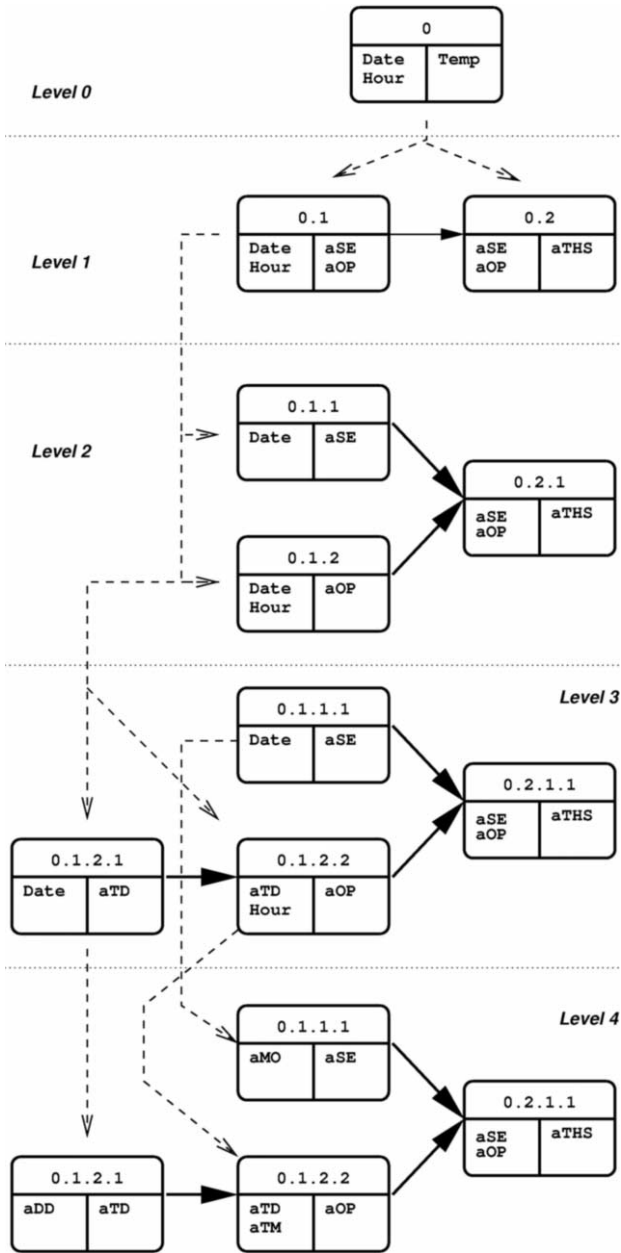


Figure 3. An Example of multilevel ARD model

3. Logical Design Using XTT

Extended Tabular Trees (XTT) [3] is a new visual knowledge representation and design method. It aims at combining some of the existing approaches such as decision-

tables and decision-trees by building a special hierarchy of Object-Attribute-Tables [5], [4]. Some ideas used in its development were previously presented in [8], [9], [10].

XTT *syntax* is based on some elementary concepts, allowing for building the hierarchy, these are: attribute, cell, header, row, table, connection, and tree. The header can be interpreted as the *table scheme*. It contains the list of attributes and their operating contexts: *conditional*, *assert*, *retract*, and *decision*.

The *visual* representation of XTT is crucial from the rule-based system design point of view. An example of an XTT structure modelling the thermostat example is shown in Figure 4.

The *semantic interpretation* of XTT is very important for the RBS design process. It uses some well-established concepts. A row of a table is interpreted as a production rule of the form: IF condition THEN conclusion. The condition part of the rule is mapped to the *Conditional context* (?) of the row. On the other hand, the conclusion part is mapped to *Assert* (+), *Retract* (-), and *Decision* (\rightarrow) contexts of the Row. The use of *Assert/Retract* contexts allows for dynamic modification of the rule-base. So in practice it is an *extended rule*, allowing for *non-monotonic* reasoning, with explicit *control* statements.

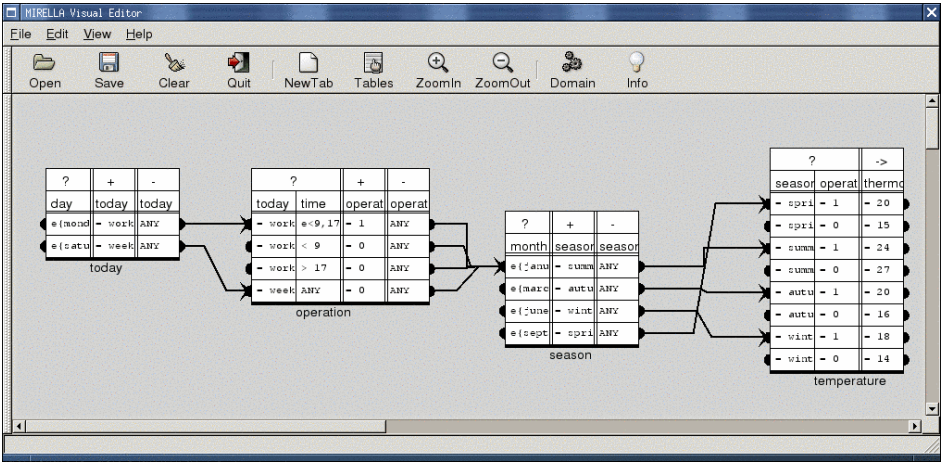


Figure 4. An example of XTT structure

An XTT table is interpreted as a set of *rules*, where *rule j + 1* is processed after *rule j*. Rules grouped in one table share the same attributes. This concept is similar to decision tables and to Relational Data Base knowledge representation.

A concept of XTT tree allows for building a hierarchy of tables. Each row *x* of a table *w* can have a right connection to another row *z* in another table *y*. Such a connection implies logical AND relation in between. Rule processing is then transferred from row *j* in table *x* to row *k* in table *y*. This concept is similar to decision trees.

An important feature of XTT is the fact that, besides its visual representation, they have a well-defined, *logical form* which may be formally analysed [4]. The XTT hierarchy can be mapped to a corresponding PROLOG code. This representation is crucial to the formal verification of *XTT*.

On the *machine readable level* XTT can be represented in an XML-based *XTTML* (*XTT Markup Language*) suitable for import and export operations, as well as translated to XML-based rule markup formats such as *RuleML*.

In order to build a working prototype of a RBS the XTT design can be automatically translated to a PROLOG-based representation.

4. Automated Implementation Using PROLOG

Transformation from XTT tables to a PROLOG-based representation allows for obtaining a *logically equivalent* code that can be executed, analysed, verified, optimised, translated to another language, transferred to other system, etc.

In order to fully represent an XTT model several issues have been solved, namely: *fact* (Object-Attribute-Value triple) representation has been chosen, attribute *domains* with all the constraints have been admitted, rule syntax has been defined, etc. Knowledge base is *separated* from the inference engine, and inference control mechanisms have to be implemented.

Every XTT cell corresponds to a certain *fact* in the rule base. A fact is represented by the following PROLOG term:

```
f (<attribute_name>, <value_type>, <value>)
```

where *attribute_name* is an XTT attribute name, *value_type* is one of {atomic, set, interval, natomic, nset, ninterval}, (last three are negated types) and *value* is the attribute value held in cell, possibly non-atomic one.

The attribute specification allows for non-atomic attribute values and expressing relational expressions through mapping to a non-atomic value sets.

The whole table-tree structure of XTT is represented by one *flat* rule-base. Every row in a table corresponds to a single rule. Every rule is encoded as a PROLOG fact of the form:

```
rule ( <TableNo.>, <RuleNo.>, <conditional>, <retract>,  
      <assert>, <decision>, <next_rule>, <else_rule>)
```

with the obvious meaning of the components, consistent with the XTT component structure. Tables are listed beginning with the *root table*. The rule-base is separated from the inference engine code. The inference process is performed by a PROLOG-based meta-interpreter (inference engine) [11]. The engine has a simple user shell providing access to its main functions.

Using the Thermostat example, the *Operation* table (see the second table in Figure 4) is represented by the PROLOG code shown below:

```
rule(2,3, [f(aTD,atomic,workday), f(aTM,interval,i(9,17))],  
         [f(aOP,atomic,_)], [f(aOP,atomic,true)], [],3,7).  
rule(2,4, [f(aTD,atomic,workday), f(aTM,interval,i(0,8))],  
         [f(aOP,atomic,_)], [f(aOP,atomic,false)], [],3,7).  
rule(2,5, [f(aTD,atomic,wd), f(aTM,interval,i(18,24))],  
         [f(aOP,atomic,_)], [f(aOP,atomic,false)], [],3,7).  
rule(2,6, [f(aTD,atomic,wk)],  
         [f(aOP,atomic,_)], [f(aOP,atomic,false)], [],3,7).
```

As a part of PROLOG implementation a framework for the analysis and verification of the XTT knowledge base has been developed. The framework allows for the implementation of different external verification and analysis modules (plugins), implemented in PROLOG. They have direct access to the system knowledge base. Each module reads the rule base and performs the analysis of the given property, e.g.: subsumption, determinism, completeness, or possible reduction.

5. CASE Tool Support

Both ARD and XTT are visual design methods. In order to fully benefit from their advanced capabilities, computer tools supporting the visual design and PROLOG-based implementation process should be developed. The creation of such tools was simplified by creating a markup, XML-based, description of ARD and XTT knowledge representation languages. So the XML description is a machine-readable encoding of the visual representation.

ARD diagrams can be described using ARDML, an XML-based language. An excerpt of ARDML code representing diagram shown in Figure 2 is shown below:

```
<ard> <table head="0.1" level="1">
  <in>
    <att type="c">Date</att> <att type="c">Hour</att></in>
  <out>
    <att type="p">aSE</att> <att type="p">aOP</att> </out>
  </table>
</ard>
```

XTT structure can be described using XML-based XTTML. An excerpt of XTTML code representing diagram shown in Figure 4 is shown below:

```
<xttml:table_list>
  <tab tid="1" parent="0" state="0">
    <label>today</label> <type>0</type>
    <header>
      <hdratt context="0">day</hdratt>
      <hdratt context="1">today</hdratt>
      <hdratt context="2">today</hdratt></header>
    <rows>
      <row rid="1" parent="1" state="0">
        <in_connect/> <out_connect>1</out_connect>
        <cells>
          <cell cid="1" parent="1" state="0">
            <att>day</att> <oper>6</oper>
            <val atomic="0" ignore="0">
              <nav lbrace="2" hbrace="2">
                <navlo>monday</navlo> <navhi>monday</navhi>
                . . . . .
              </nav>
            </val>
          </cell>
        </cells>
      </row>
    </rows>
  </tab>
</xttml:table_list>
```

Both XML formats may be used to exchange information between visual case tools supporting the ARD/XTT design. They can also be directly transformed into other XML formats using XSLT (transformation language). The XTTML has been used as the foundation of Mirella CASE tool.

MIRELLA [3], [12] is a prototype CASE tool for the XTT visual design method. Its main features are:

- support for visual design methodology, using XTT,
- support for integrated, incremental design and implementation process,
- possibility of the on-line, incremental, local verification of formal properties,
- direct translation of system design into PROLOG-based executable prototype, so that system operational semantics is well-defined, and
- use of XML-based flexible data exchange formats.

The XTT structure shown in Figure 4 has been designed in MIRELLA. The corresponding PROLOG prototype has been automatically generated and analysed. The example RBS system (see Section 2) was proven to have basic properties such as determinism, completeness and lack of subsumption. However, MIRELLA detected a possible rule reduction:

```
?- vpr(4).
Rule: 4.11 may be glued with rule: 4.15, reduced fact: f(aSE,set,[spr,aut])
Rule: 4.15 may be glued with rule: 4.11, reduced fact: f(aSE,set,[aut,spr])
No more reduction of rules in table 4
```

Two rules in the fourth ("temperature") table (see Figure 4) could be substituted by a single rule with use of a non-atomic attribute value:

```
rule(4,11,[f(aSE,set,[aut,spr]),f(aOP,atomic,yes)],
      [],[],[f(aTHS,atomic,20)],0,_).
```

Currently, the prototype version of MIRELLA⁴ supports the visual XTT design with possibility of specifying full logical structure of the RBS. The PROLOG prototype generation and verification is fully supported as well. However, conceptual design is limited to attribute list and domain specification. The incorporation of ARD conceptual design is planned in the future. Using the modular architecture of Mirella it is also possible to incorporate the ARD editor as an external module.

6. Concluding Remarks

The paper outlines elements of a meta-level approach to integrated RBS modelling, design, and implementation process, with top-down hierarchical design methodology, including three phases: conceptual (ARD), logical (XTT), physical (PROLOG), providing clear separation of logical and physical design phases. It offers equivalence of logical design specification and prototype implementation.

The ARD/XTT design and implementation process can be supported by visual CASE tools, such as open source-based MIRELLA. Future work on ARD, XTT, and MIRELLA will include full CASE tool support for the ARD method, as well as purely XML-based XTT to PROLOG transformation.

⁴ For current Mirella-related developments see project page mirella.ia.agh.edu.pl

References

- [1] Liebowitz, J., ed.: The Handbook of Applied Expert Systems. *CRC Press*, Boca Raton (1998)
- [2] Hopgood, A.A.: Intelligent Systems for Engineers and Scientists. 2nd edn. *CRC Press*, Boca Raton London New York Washington, D.C. (2001)
- [3] Nalepa, G.J.: Meta-Level Approach to Integrated Process of Design and Implementation of Rule-Based Systems. PhD thesis, AGH University of Science and Technology, AGH Institute of Automatics, Cracow, Poland (2004)
- [4] Ligeza, A.: Logical Foundations for Rule-Based Systems. *Uczelniane Wydawnictwa Naukowo-Dydaktyczne AGH w Krakowie*, Kraków (2005)
- [5] Ligeza, A.: Logical support for design of rule-based systems. reliability and quality issues. In Rousset, M., ed.: ECAI-96 Workshop on Validation, Verification and Refinement of Knowledge-based Systems. Volume W2. *ECAI'96*, Budapest (1996)
- [6] Connolly, T., Begg, C., Strachan, A.: Database Systems, A Practical Approach to Design, Implementation, and Management. 2nd edn. *Addison-Wesley* (1999)
- [7] Negnevitsky, M.: Artificial Intelligence. A Guide to Intelligent Systems. *Addison-Wesley*, Harlow, England; London; New York (2002)
- [8] Ligeza, A., Wojnicki, I., Nalepa, G.: Tab-trees: a case tool for design of extended tabular systems. In et al., H.M., ed.: Database and Expert Systems Applications. Volume LNCS 2113 of Lecture Notes in Computer Sciences. *Springer-Verlag*, Berlin (2001) 422-431
- [9] Nalepa, G.J., Ligeza, A.: Designing reliable web security systems using rule-based systems approach. In Menasalvas, E., Segovia, J., Szczepaniak, P.S., eds.: Advances in Web Intelligence. AWIC 2003, Madrid, Spain, May 5-6, 2003. Volume LNAI 2663., Berlin, Heidelberg, New York, *Springer-Verlag* (2003) 124_133
- [10] Nalepa, G.J., Ligeza, A.: Security systems design and analysis using an integrated rule-based systems approach. In Szczepaniak, P.S., Kacprzyk, J., Niewiadomski, A., eds.: Advances in Web Intelligence: 3rd international Atlantic Web Intelligence Conference AWIC 2005: Lodz, Poland, June 6-9, 2005. Volume LNAI 3528 of Lecture Notes in Artificial Intelligence., Berlin, Heidelberg, New York, *Springer-Verlag* (2005) 334_340
- [11] Covington, M.A., Nute, D., Vellino, A.: Prolog programming in depth. *Prentice-Hall* (1997)
- [12] Nalepa, G.J., Ligeza, A.: A graphical tabular model for rule-based logic programming and verification. In Bubnicki, Z., Grzech, A., eds.: Proceedings of 15th International Conference on Systems Science, Wrocław, Wrocław University of Technology, *Oficyna Wydawnicza Politechniki Wrocławskiej* (2004)

Model Management Based on a Visual Transformation Language

Michał ŚMIAŁEK ^{a, b} and Andrzej KARDAS ^a

^a *Warsaw University of Technology,*

^b *Infovide S.A.,*

Warsaw, Poland

e-mail: smialek@iem.pw.edu.pl

Abstract. Modeling becomes the most prominent method of dealing with software complexity. Models for software systems can be created on different levels of abstraction and in different phases of the software development process. In order to manage these various models effectively we need to apply methods for keeping them coherent and traceable. The current paper proposes a method for defining and executing transformations between models written in the currently most popular modeling language (UML). We propose a general purpose transformation language that allows for specifying transformation mappings between different models. This language has a visual (MOF-based) notation and allows for defining transformation templates and rules. With this language, the software developers are capable of determining traceability links between individual model elements. Moreover, the transformations can be executed automatically to high extent, with the use of a dedicated tool that extends a standard UML CASE tool.

Introduction

Complexity of contemporary software systems is constantly growing. In order to deal with this complexity effectively, appropriate abstraction techniques are needed. Creating models of software, slowly becomes the most prominent method for dealing with this complexity and realizing abstraction. Models allow for managing the structure and dynamics of even the most complex systems on different levels of abstraction, making it possible to reveal only the amount of detail that is comprehensible for the model readers.

Currently, the most widely used software development methodologies are based on the object oriented paradigm. After an initial boom in the area of software modeling methodologies based on object orientation, current tendency is to unify different approaches. The most prominent example in this area is the Unified Modeling Language (UML) [1]. UML is already a standard used by vast majority of companies in the software industry and managed by an independent standardization body – the Object Management Group (OMG). UML in its current version 2.0 offers thirteen diagram types that enable showing various aspects of software and associated problem (business) domains on different levels of abstraction. Development of the UML standard goes in the direction of adding precise semantics to its diagram elements. Defining precise semantics allows for bringing closer the idea of “visual programming”, where coding would mean drawing diagrams instead of writing textual code.

The most significant problem in using UML is to organize diagrams into models that would form a clear path from user requirements, through architectural design to detailed design and code. Taking this into account, the OMG has introduced a new modeling framework, called Model Driven Architecture (MDA) [2], [3]. MDA is based on the idea of model transformation. According to this idea, models close to the business domain (Computation Independent Models) would get transformed into abstract architectural models of the system, independent of the programming platform (Platform Independent Models). These models could then be transformed into models dependent on the software platform (Platform-Specific Models), and then directly translated into working code. Such a clear path, from visual, model based requirements, through design models to code could be partially automated, thus giving faster development cycles (see e.g. [4], [5]).

In order to automate model transformation we need a transformation language that would allow for precise mapping of one type of models into another type of models [6]. An appropriate request for proposal has been issued by the OMG [7]. In reply to this proposal, several transformation languages have already been proposed (see e.g. [8], [9], [10] and [11] for summary). However, most of the propositions concentrate on PIM to PSM mappings. In the current paper we propose a transformation language that is suitable also for mappings on the CIM level. This language is based on visual notation that allows for easier definition of model mappings. The paper also presents a tool that verifies applicability of the proposed language to transforming models stored in general CASE tool repositories.

1. Model Transformations Based on UML

The key concept of MDA is the definition of model transformation [12]. Having a standard modeling language, as UML, we have to define a way to translate one type of model into another type of model. This method would have to transform individual elements and relationships (like: classes, use cases, associations, messages, etc.) in one of the models into separate elements and relationships in the other model. We need to define this method and execute it on the source model in order to obtain automatically (or semi-automatically) the target model.

The method to be applied when transforming models usually depends on the needs of the model creators. For this reason, we need to develop a language that would allow defining different methods for model transformation. This language would offer capabilities to specify appropriate rules for mapping source elements into target elements. One of the method for specifying these rules is to define appropriate UML profiles (see e.g. [13]). Another method is to define a meta-model of transformation in terms of OMG's MOF [14] specification. This approach would be compatible with MOF-based definition of UML itself [1].

The MOF approach gives us the advantage of creating a formal specification for our transformation language by using this standard language-definition language (meta-language). In the meta-modeling sense, this specification is on the "meta" level, which is illustrated on Figure 1. With MOF, we can define an abstract language that possesses all the needed syntactic and semantic well-formedness rules. By adding to this abstract language also certain concrete syntactic notations, we receive a complete concrete, visual transformation language (VTL). This concrete language can be used by the

model developers to determine model transformation rules. These rules can be used to perform actual transformations on the UML modeling level.

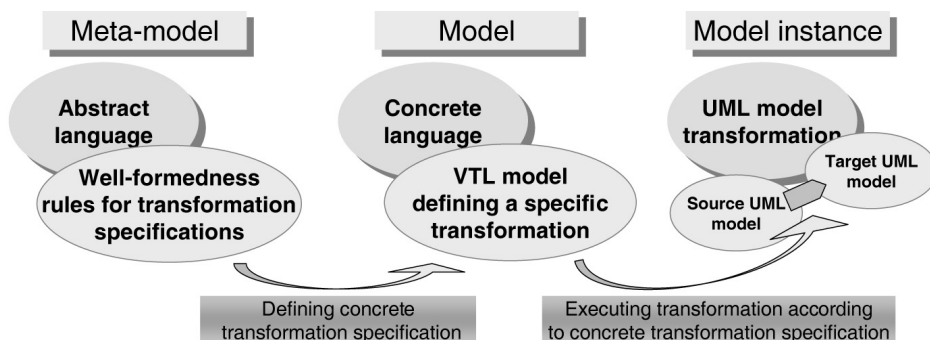


Figure 1. Model transformation language architecture based on meta-modeling layers

In the following two sections we shall present the proposed visual transformation language (VTL). We define the language's abstract syntax by defining its MOF-compliant metamodel. We then specify the concrete syntax by giving an example of the proposed visual notation.

2. Visual Transformation Language – Abstract Syntax

The source and target elements of our transformations represent appropriate elements found in UML. We normally want to transform classes, interfaces, use cases, actors, components, etc. Such elements can be connected through various relationships: associations, dependencies, generalizations, etc. In order to perform a transformation on such a model we have to find an appropriate pattern, composed of source elements and relationships between them. Having such a pattern we can apply certain transformation rules. These rules determine the actual mapping from source elements and relationships into target elements and relationships. Our transformation language should, thus, be capable of defining templates for patterns in the source model, rules that describe mapping of elements consistent with these templates, and templates for relationships in the target model.

The above requirements for our language are illustrated on Figure 2. The language would be used to define transformation specifications. This model would contain source templates, transformation rules and target templates. The transformation model can be used to perform actual transformations (transformation instances). If one of the source template matches a fragment in the source model, appropriate transformation rules are applied in order to obtain a transformed fragment in the target model. This target fragment is consistent with an appropriate target template.

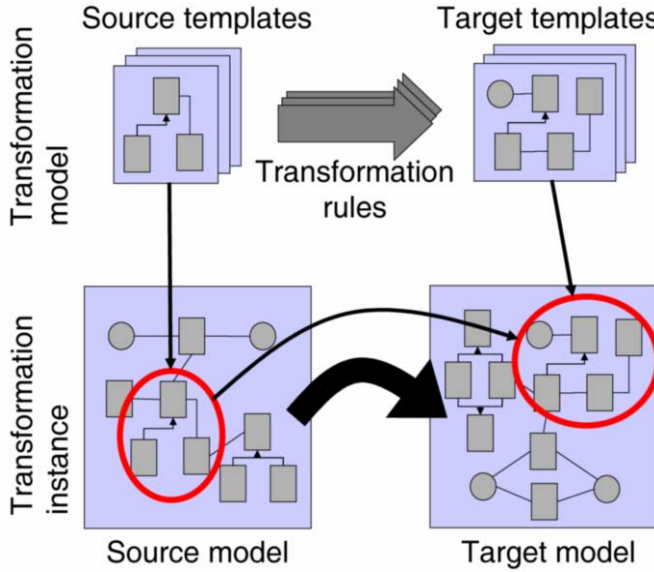


Figure 2. Transformation rules and their application

Every transformation defined in our language, in order to be complete, has to include the following information:

- definition of the source elements of the transformation
- definition of the target elements of the transformation
- mapping rules that control the translation of the source model elements into the target model elements

This is reflected in the metamodel of our transformation language, shown on Figure 3. Every Transformation is composed of several source and target ComplexElementTemplates and several ComplexTransformationRules. It can be noted that all these elements of our metamodel are derived from the Package metaclass found in the UML specification (in the Kernel package) [1]. Basic elements of every Transformation are TransformationElements that reflect elements in the source and target models of a transformation instance. These TransformationElements participate in SimpleTransformationRules and SimpleElementTemplates. Every simple rule is composed of an appropriate link (TransformationLink or TemplateLink) that connects two TransformationElements.

Figure 4 shows important details of individual elements in our metamodel. Every TransformationElement has its type. This type is defined with the ElementTypes enumeration, which can have values reflecting various metaclasses found in the UML metamodel (like: Class, UseCase, Interface and so on). The transformationSide can be either source or target. This determines, whether the element can participate in source templates or in target templates. An important attribute of the TransformationElement metaclass is the stereotypeName. The stereotype allows for performing more fine-grained transformations than only based on element types. By setting the value of this attribute in a transformation element, the transformation

developer can determine differences in transforming model elements with different stereotypes. Another important meta-attribute of `TransformationElement` is `packageName`. The value of this attribute specifies the package in the source model where the element should be sought for or the package in the target model where the element should be placed in.

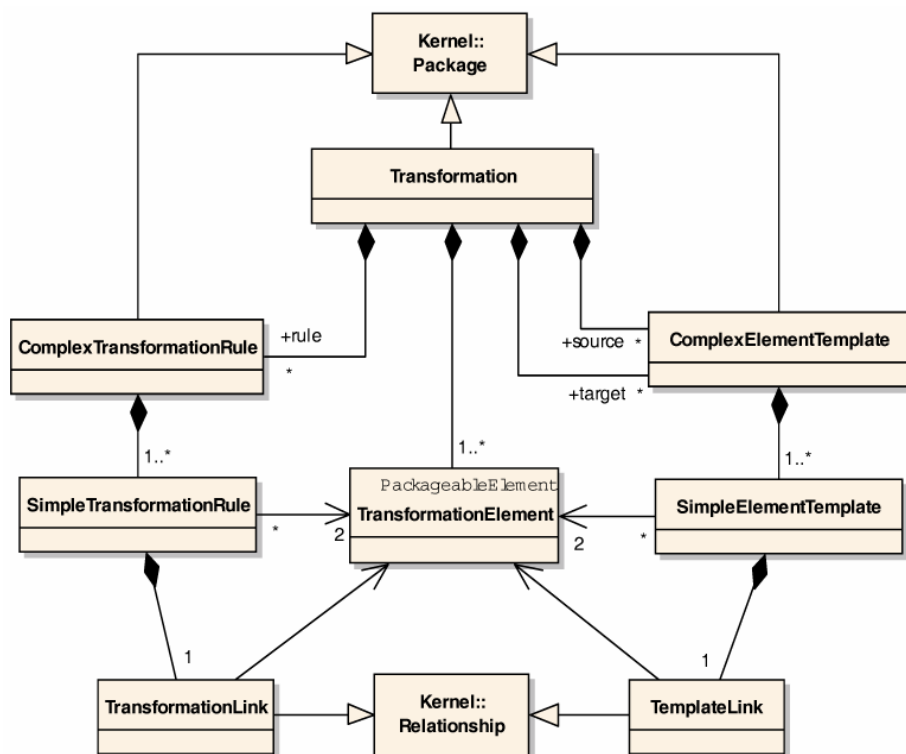


Figure 3. Metamodel of the transformation

The remaining two meta-attributes of `TransformationElement` define the type of name conversion and member conversion respectively. Name conversion is very important, as it determines how the name of the current element will be changed in the element on the other side of an appropriate `TransformationLink`. One possibility is just to copy the name. Another two possibilities include adding a prefix or a postfix. The resulting name can be also set by the developer “by hand”. Member conversion for a transformation element specifies how the elements members (like attributes, parts or operations) will be converted in the other element. Here we will mention two possibilities: we can copy all the members or we can suppress copying (no members copied).

Name and member conversions are performed on elements (only between a source and a target element) connected with `TransformationLinks`. It can be noted that these links are directed. The direction determines whether this particular transformation can be performed between source and target only or in both directions.

We can also observe, that the TransformationLink has the same conversion attributes as TransformationElement. This is due to the fact that a single element can be converted to several other elements. In such situation each TransformationLink coming from a source element can add its own control over conversion in addition to standard conversion specified inside this element.

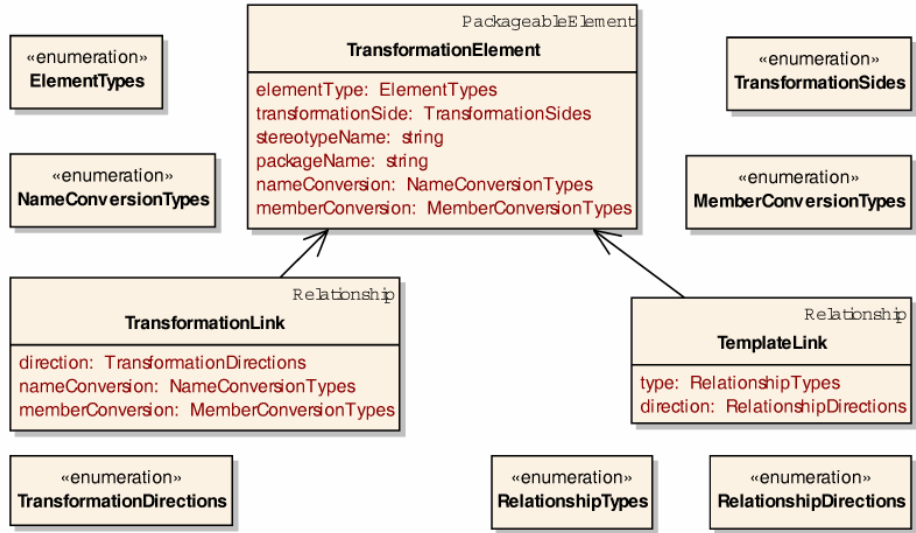


Figure 4. Definition of metamodel elements

For the definition of transformation to be complete, we also need to define the TemplateLink. This connector determines relationships between elements in the source or target templates. TemplateLinks can have types that reflect metaclasses of the UML specification that derive from the Relationship metaclass. Thus, we can have TemplateLinks typed e.g. as Dependencies, Associations or Generalizations. TemplateLinks can connect only two TransformationElements on the same transformationSide. This allows for defining templates for the source or target models.

3. Visual Transformation Language – Concrete Syntax

We shall describe the concrete syntax of our Visual Transformation Language by using an example shown on Figures 5-7. These diagrams contain a VTL model (in consistence with what we said in Section 1) of a specific transformation, compliant with the meta-model described in the previous section.

As it can be noted, VTL TransformationElements are denoted with a symbol identical to the UML’s class symbol. Every such symbol is named (as being derived from a UML NamedElement, see [1], Fig. 4 on p. 24). All the meta-attributes of TransformationElements are simply represented as attributes of transformation

model elements with assigned specific values. TemplateLinks are denoted with appropriate connectors whose notation is derived from appropriate UML's relationships (see Figures 5 and 6). TransformationLinks are denoted as associations with an added constraint that reflects values on the "conversion" meta-attributes.

Figure 5 shows the source template of our transformation model. As we can see, the transformation is performed, whenever a generalization relationship between two classes is found. Moreover, these two classes should be stereotyped as <<userInput>>. The transformation is independent of the package in which the template elements are found (packageName = ?). Default name conversion for both of the classes is to add a prefix, and for their members – to copy them into the target elements.

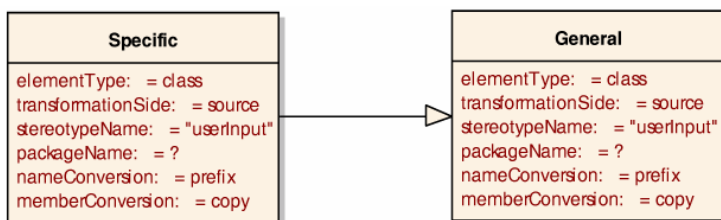


Figure 5. Example source template

Source model fragments that match template shown on Figure 5 will get transformed into model fragments matching template on Figure 6. This target template contains four TransformationElements representing four classes in the target model. After transformation, these classes will be related with a generalization, two aggregations and two associations. Their stereotypes will be set – according to the template – to <<dialogData>>, <<XMLPacket>> and <<dialogWindow>>. These classes will be placed in appropriate packages (User Interface, Application Logic and Interfaces). It can be noted that name and member conversion is not defined, which suggests that the transformation is uni-directional (only from source to target).

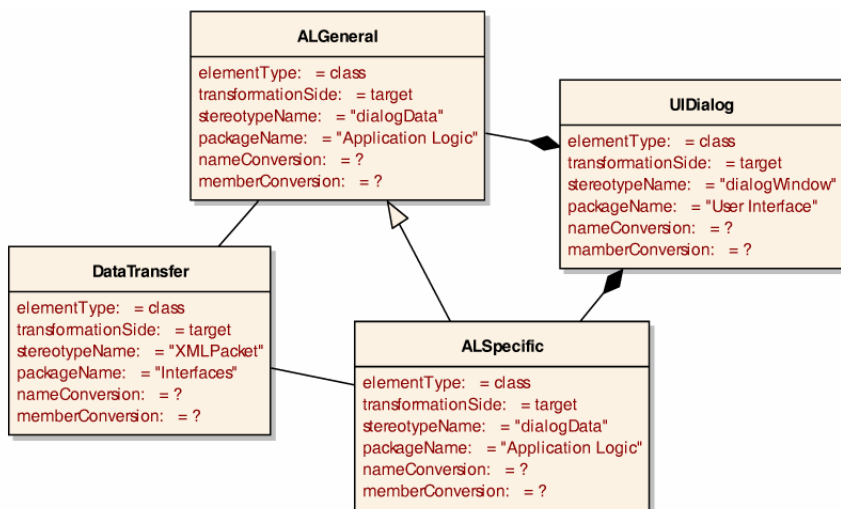


Figure 6. Example target template

The transformation model is completed with transformation rules, shown on Figure 7. These rules determine which source elements will get converted into which target elements. The arrows show direction of this conversion (from source to target). Some TransformationLinks are adorned with constraints. These constraints override standard conversion rules defined in appropriate source TransformationElements. As we can see, the UIDialog target elements will have names converted from the Specific source elements only (with a prefix, according to Figure 5). No members will be copied into UIDialog elements. On the other hand, the DataTransfer elements will have member lists being a concatenation of members from General and Specific elements.

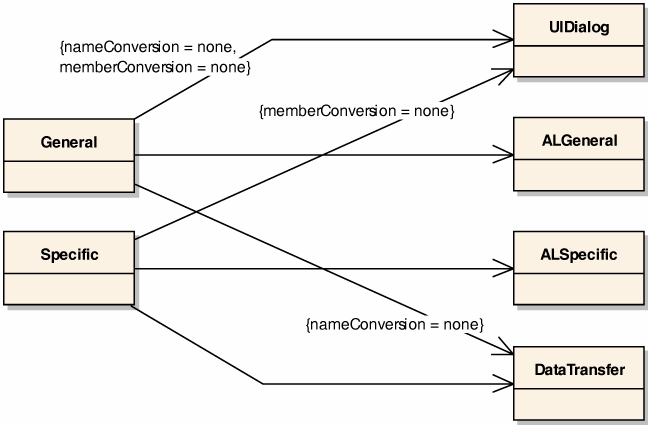


Figure 7. Example transformation rules

The above described transformation model is the basis for performing the actual transformation on UML models. An example of such a model is shown on Figure 8. This model contains three classes stereotyped exactly as specified in the source transformation template. In this model we have two fragments that match correctly the template from Figure 5. It can be noted that these two fragments have a common General class – the UserData class.

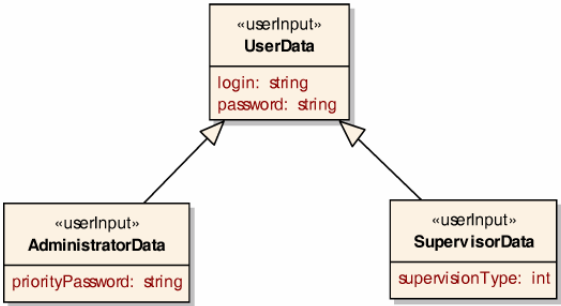


Figure 8. Example source model ready to be transformed into a target model

After performing a transformation, the source model gets transformed into model shown on Figure 9. This diagram contains two fragments that are derived from the

template on Figure 6. These two fragments are again joined by one of the classes (the CUserData class). The joining class is transformed from the joining class in the source model. It can be noted, that the class name has been extended with a prefix (letter 'C'). Both attributes of UserData have been copied into CUserData. Attributes from UserData have also been copied into XAdministratorData and XSupervisorData. These two classes have also attributes copied from two other classes from the source model. This is consistent with the transformation rules shown on Figure 7.

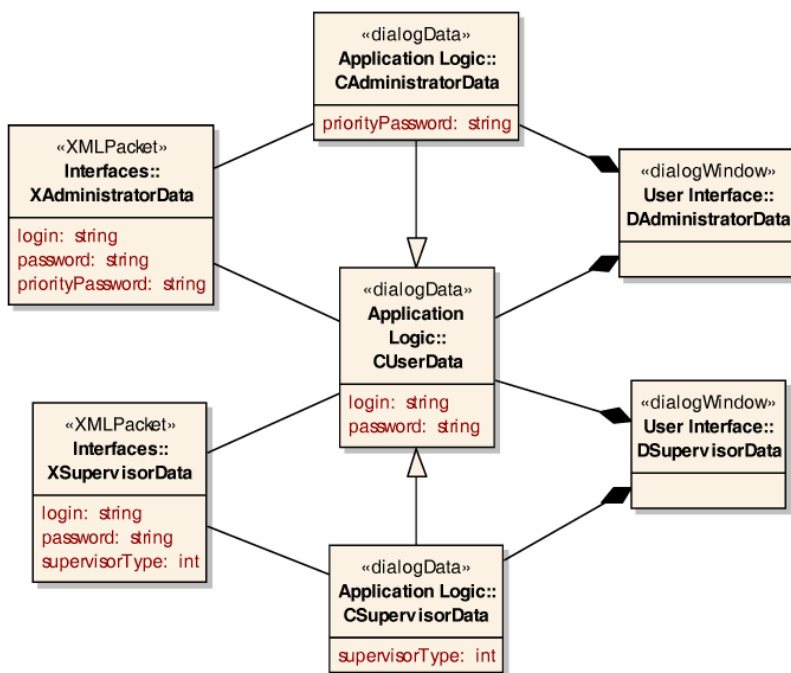


Figure 9. Example target model transformed from the source model according to specified rules

The above example shows a very simple transformation of an analytical model into a design model. Three analytical classes are translated into seven design classes placed in different architectural layers (user interface, application logic, data transfer). This gets reflected in placing the resulting classes in three different packages that are shown as namespace prefixes in the class names.

4. Model Management with a Transformation Tool

It is obvious that VTL specifications are meant for automatic transformations. Only an appropriate tool can effectively perform more complex transformations relieving developers from tedious copying and mapping classes between analytical and design packages. Figure 10 shows a typical task for a developer: take a model placed in a package on the analytical level and create a model placed in several packages on the design level. This second model could then be generated directly into code.

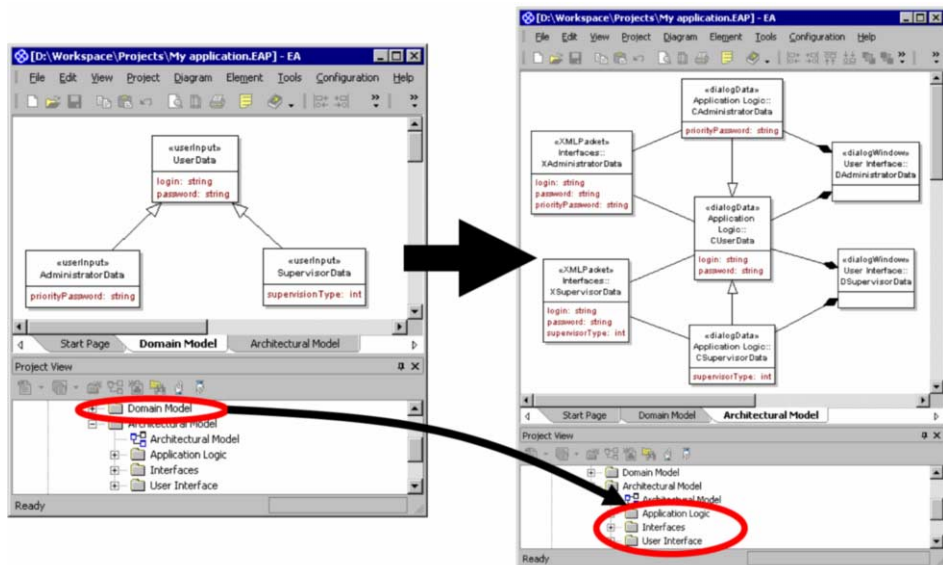


Figure 10. Transformation in a CASE tool

This task can be partially automated if we can define standard transformation rules. With such rules present, a tool can apply these rules and perform appropriate transformation. Of course, the developer still would need to maintain the target model and add appropriate new attributes or operations to generated classes.

Figure 11 shows an application of a tool developed as a plug in for a standard UML CASE tool¹. The plug in allows to prepare a model of transformation written in VTL. Being an extension to UML, this model is stored inside the standard repository of the UML CASE tool. After defining all the source and target templates, and transformation rules, developer can invoke the transformation procedure. This procedure is quite simple, as it involves only choosing the source and target model packages. Additionally, the developer has to assign prefixes and postfixes to appropriate element transformations. After this, the plug-in tool takes the source model and matches it with source templates according to the transformation model. For each of the fragments in the source model that matches a template, appropriate transformation into the target model is performed.

5. Conclusions and Directions for Future Work

Solving the problem of getting efficiently from user requirements to code seems to be crucial for the present software developers. Experience from various projects and case studies shows that this path can be significantly improved with the use of visual modeling techniques. With the proposed visual transformation language we receive another link in this modeling process. The language offers capability to define standard paths from the initial models that define the user needs, through architectural

¹ The tool used here is Enterprise Architect from SparxSystems which offers extensive and comprehensive programmers interface.

frameworks up to design solutions that can lead directly to code. It is very important that the language uses a very similar modeling notation as the models that are subject to transformations. This allows the developers to build visual templates for their transformations just as they would develop the actual models themselves.

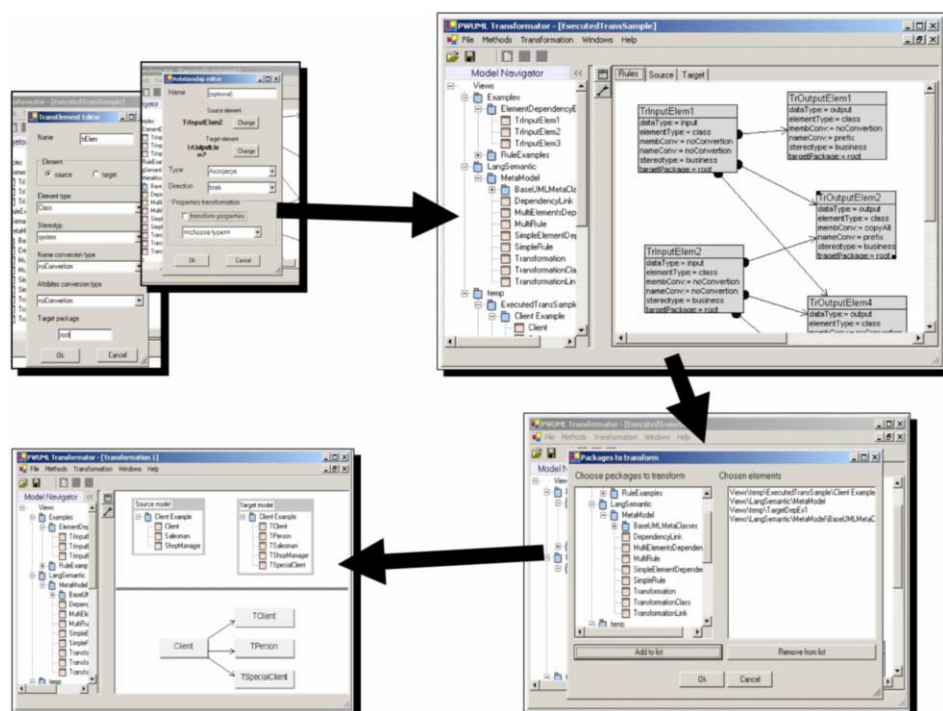


Figure 11. Illustration of transformation lifecycle

Using MOF as a meta-language for VTL, and treating it as an extension of UML, gives us an advantage of the new language being compatible with the existing UML 2.0 CASE tools. It was relatively easy to use the existing repository structure of the chosen CASE tool (i.e. the Enterprise Architect), to store the transformation definition together with the transformed models. This also allows for easy application of the same transformation to different models simply by copying appropriate package with the transformation definition.

It has to be stressed that the proposed language does not solve all the problems associated with MDA/MDD [15] transformations. The target models still need a significant amount of human effort in order to accommodate them to appropriate technologies or architectural solutions. The future development of the VTL language, together with the transformation tool should go in the direction of extending the transformation rules into more detailed areas of particular UML models. With more detailed definition capabilities, most of the development effort could go into defining transformations. These could be defined only once for all applications in the given

problem domain or using the same technology. With these transformations in place, the majority of development would be concentrated on creating requirements consistent with the **real** user needs. The rest could be performed almost automatically – with a little intervention of the developers. However, this is still to come...

References

- [1] Object Management Group: Unified Modeling Language: Superstructure, version 2.0, Revised Final Adopted Specification, ptc/04-10-02. (2004)
- [2] Miller, J., Mukerji, J., eds.: MDA Guide Version 1.0.1, omg/03-06-01. Object Management Group (2003)
- [3] Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model Driven Architecture. Addison-Wesley (2004)
- [4] Berkem, B.: How to increase your business reactivity with UML/MDA. *Journal of Object Technology* 2 (2003) 117-138
- [5] Orr, K., Hester, R.: Beyond MDA 1.0: executable business processes from concept to code. *MDA Journal* 11 (2004) 2-7
- [6] Bezivin, J., Gerbe, O.: Towards a precise definition of the OMG/MDA framework. In: *Proc. 16th Annual International Conference on Automated Software Engineering ASE 2001*. (2001) 273_280
- [7] Object Management Group: MOF 2.0 Query / Views / Transformations RFP, ad/2002-04-10. (2002)
- [8] Koehler, J., Hauser, R., Kapoor, S., Wu, F.Y., Kumaran, S.: A model-driven transformation method. In: *Proc. Seventh IEEE International Enterprise Distributed Object Computing Conference EDOC 2003*. (2003) 186-197
- [9] Duddy, K., Gerber, A., Lawley, M.J., Raymond, K., J, S.: Model transformation: A declarative, reusable patterns approach. In: *Proc. 7th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2003*, Brisbane, Australia (2003) 174-195
- [10] Kalnins, A., Barzdins, J., Celms, E.: Basics of model transformation language MOLA. In: *Workshop on Model Driven Development (WMDD 2004)*. (2004)
- [11] Czarnecki, K., Hensen, S.: Classification of model transformation approaches. In: *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*. (2003)
- [12] Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The missing link of MDA. *Lecture Notes in Computer Science* 2505 (2002) 90-105
- [13] Śmiałek, M.: Profile suite for model transformations on the computation independent level. *Lecture Notes on Computer Science* 3297 (2005) 269-272
- [14] Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification, Final Adopted Specification, ptc/03-10-04. (2003)
- [15] Aagedal, J.O., Bezivin, J., Linington, P.F.: Model-driven development. In Malenfant, J., Ostvold, B.M., eds.: ECOOP 2004 Workshop Reader. Volume 3344 of LNCS. Springer-Verlag (2005)

Exploring Bad Code Smells Dependencies

Błażej PIETRZAK and Bartosz WALTER

Institute of Computing Science,

Poznań University of Technology, Poland

e-mails: {Blazej.Pietrzak, Bartosz.Walter}@cs.put.poznan.pl

Abstract. Code smells reveal in different ways: with code metrics, existence of particular elements in an abstract syntax tree, specific code behavior or subsequent changes in the code. One of the least explored smell indicators is the presence of other smells. In the paper we discuss and analyze different relations existing among smells and suggest how to exploit this information in order to facilitate discovering of other smells. The considerations are completed with analysis of a Large Class smell and its related odors within a popular open source project codebase.

Keywords. Refactoring, code smells

Introduction

The source code quality is one of the factors affecting the development and maintenance cost [1]. The low quality results in more frequent bugs and, consequently, higher expenditure on software evolution-related activities [2]. On the other hand, the care for quality is costly, yet giving a chance for a savings on further software maintenance.

Refactoring is an XP-originated technique that attempts to reconcile the goals [3]. It is a transformation of source code that improves its readability and structure, while preserving its behavior [4]. The XP software product life cycle, unlike the classical one, focuses on the maintenance phase, and refactoring is its primary practice to preserve the source code quality. Therefore, if performed frequently, refactoring helps in keeping the code healthy.

Prior the code is refactored, there is a need for diagnosing the problem and suggesting treatment methods. The notion of bad code smells, introduced by K. Beck, gives a vague and deliberately ambiguous indication of the source code flaws that could possibly lead to an erroneous execution and require refactoring. There are over 20 different smells defined [4], each representing a separate violation: from overuse of temporary fields, ending up with misplaced classes within the inheritance hierarchy. Despite of that, the term *smell* is a common umbrella for various structural or behavioral source code faults. As a result, there is no single method for smell detection and identification. Each smell reveals itself in a unique way by a set of distinctive symptoms.

To capture the volatile nature of smells, in [5] we proposed a holistic model of odor detection. We identified six data sources that provide information about smells, which have to be combined together to obtain more accurate findings. They include commonly applied indicators, like metrics or AST elements, but also human intuition

and unit tests. We also noticed that existence of some smells gives some hints concerning other flaws. The information either suggests or excludes the presence of other smells or, when combined with other symptoms, provides a clearer view of a smell, that otherwise could be too faint and, effectively, overlooked.

In this paper we analyze the relations that exist among different smells and how they could be exploited for more effective smell detection. We also discuss the example of a *Large Class* smell and its related smells, and how *Large Classes* identification can be supported by the knowledge of other odors presence.

The paper is structured as follows. In Section 1 we shortly summarize the research on code smells and the data sources of smell symptoms we identified. We also stress the importance of re-using the already detected smells for identification of other ones. Section 2 presents various relations existing among smells and discusses how they can be exploited in detection of other smells. They are also compared to associative rules applied in data mining. In Section 3 we analyze the relations of the *Large Class* smell with other odors and how they support detection of this smell. In Section 4 we evaluate the findings from Section 3 on selected classes taken from Jakarta Tomcat project [6], and finally provide some concluding remarks in Section 5.

1. Bad Code Smells and their Indicators

Refactoring, like other health-preserving and restoring activities, requires a precise notion of what the health means. Unfortunately, there is no generally accepted definition of code health. The term *bad code smell*, coined by K. Beck [4], is used among XP-adopters for describing the unhealthy code. According to it, smells are structures in the code that “suggest (sometimes scream for) the possibility of refactoring” [4]. This definition, mostly based on a human sense of aesthetics, stands in contrast to the variety of individual smells that actually have been identified.

This deliberate imprecision drives also to significant problems with automated detection and identification of smells, and, consequently, refactoring. This is illustrated by the list of 20 bad smells presented by Fowler, which contains items differing in complexity, intensity and importance. Similarly, the range of code elements affected by them spans from entire class groups or hierarchies (*Parallel Inheritance Hierarchies*, *Message Chain*), through classes and objects (*Feature Envy*, *Divergent Change*, *Large Class*), then methods (*Extract Method*, *Long Parameter List*), ending up with individual variables (*Primitive Obsession*, *Temporary Field*). As a result, there exists no general method of smell detection, and each smell deserves a separate treatment.

In order to overcome this disadvantage, in [5] we proposed a multi-criteria approach to smell detection. It still does not provide a single detection method, but describes them in a uniform way. The approach originated from a conclusion that every single smell discloses multiple symptoms that need to be combined together. To reflect the programmer preferences concerning the smell importance, we adopted UTA method, which learns from the initial ranking of smells, and then applies the order to a set of variants. Therefore, it simultaneously attains two goals: balances multiple smell indicators and reflects the programmer’s personal sense of smell. A similar solution has been proposed by Moonen and van Emden [7].

We also identified and described six sources of data useful for smell detection:

- Programmer's intuition and experience,
- Metrics values,
- Analysis of a source code syntax tree,
- History of changes made in code,
- Dynamic behavior of code,
- Existence of other smells.

The former four sources derive the information directly from code or its dependents, either statically or dynamically. The last one reuses the already discovered smells, so that they can be exploited again in further discovery. Most smells co-exist with others, which means that presence of one smell may suggest presence of the related ones. This is due the common origins of some smells: a single code blunder gives rise to many design faults, and the resulting smells are not independent of each other. For example, *Long Methods* often contain *Switch Statements* that decide on the control flow, or have *Long Parameter Lists*. The latter case indicates also the *Feature Envy*, since most of data must be delivered from outside. Therefore, if a method is considered long, the danger of *Long Parameter List* also gets higher.

Consequently, it is possible to detect bad smells utilizing the already available information that could be extended, supported or rejected by the previously discovered facts.

2. Inter-Smell Relations

Even a superficial analysis of bad smells presence indicates that some of them usually appear in groups, and that the presence of a particular smell bears information about others. It is Fowler who suggests the existence of several inter-smell relations: “When a class is trying to do too much, it often shows up as too many instance variables. When a class has too many instance variables, duplicated code cannot be far behind” [4]. The nature of relations may vary: some smells originate from a common flaw; others reveal similar symptoms or can be eliminated with a single transformation. Therefore the relationship depends on the point of view and on how they could be utilized. In this paper we explore the relations that allow for effective discovering existence of other smells and, subsequently, their elimination.

Identification of smell relations reminds the discovery of associative rules in data sets [8]. The associative rules describe the relations among data items within a single transaction in terms of two attributes:

- **support ratio s** , which describes the number of transactions containing any of items participating in the rule, relative to the number of all transactions, and
- **confidence ratio c** , which is the number of transactions which actually fulfill the rule, relative to the number of supporting transactions.

Support ratio for an inter-smell relation denotes its importance and describes how many code elements the relation could actually affect. The confidence is interpreted as the relative number of relations conforming to the rule, and tells how much it can be trusted. Thus, the rules are not strict implications, but rather suggestions that a relation exists with certain probability, and may alter along with the investigated data set. The

same applies to smell detection: actual support and confidence levels may vary depending on the different factors, e.g. programming style and technology used.

It is important to notice that in order to apply the metaphor of associative rules to smell detection, the smells existence needs formalization. In a common sense the smell's primary characteristic is its intensity, whereas we need a Boolean function: a smell either exists or does not. However, it can be easily achieved by defining a minimum intensity required for the smell to exist.

There are several relations that connect smells, depending on their purpose. We identified five relations among smells that can be effectively exploited in detecting and identification of other smells:

- Simple compatibility,
- Mutual compatibility,
- Transitive compatibility,
- Aggregate compatibility,
- Incompatibility.

2.1. Simple Compatibility

Smell compatibility, denoted $A \Rightarrow B$, is the primary and simplest relation that may be identified within code. A smell B is compatible with A if the existence of A implies the existence of B with a confidence level higher than assumed. In other words, B is a companion smell of A , and usually the items bearing A are also infected with B . In many cases A is a relatively simple smell with few symptoms, and B is a more general one, with complex nature and various aspects. Sometimes the smells are mutually compatible (see Section 2.2), but the relation is basically unidirectional and does not imply the opposite connection.

As an example let us consider the *Switch Statements* smell. An infected routine is usually also a *Long Method*, since it takes different actions depending on circumstances. The opposite relation in this case, however, is not true: there are many reasons why methods get long, and the conditionals presence is only one of them.

2.2. Mutual Compatibility

Mutual compatibility is the symmetric closure of the compatibility relation: both smells are compatible with its counterpart. It is not only equivalent to two simple compatibility relations, but it also suggests that the smells share common origins and are consequences of a single code flaw. Removing the flaw may result in disappearance or reduction of intensity of both smells.

2.3. Aggregate Compatibility

Aggregate compatibility generalizes the simple compatibility relation to a case of multiple source smells. A finite set of existing smells A_1, A_2, \dots, A_n is compatible as an aggregate with a smell B if it supports the existence of the smell B with higher confidence than any of individual smells A_i . In other words, it is the synergy of several source smells that alleviates the detection of the target odor.

For example, the presence of several *Switch Statements* suggests that a routine is also considered a *Long Method*. An over-commented member function probably also performs multiple activities, which subsequently may indicate a *Long Method*. However, they separately do not imply the smell existence, because in certain circumstances such constructs are justified. But simultaneous presence of both smells raises the intensity of the smell to a higher level than individual ones.

Aggregating the smells is therefore a stronger premise for existence of some smells, but also more difficult to explore. The source smells may also depend one on another, which could affect the confidence level of the rule.

2.4. Transitive Compatibility

The relation is a specific example of aggregate compatibility with source smells depending on each other. Provided that there exist compatibility relations $p: A \Rightarrow B$ and $q: B \Rightarrow C$, we can deduce the presence of a relation $r: A \Rightarrow C$. The confidence level of r is usually equal to the product of source confidence ratios (lower than for individual smells), but it can be used as an additional indicator suggesting the existence of the target smell C .

2.5. Incompatibility

Incompatibility yields the opposite information: a smell B is incompatible with smell A , if the presence of A effectively excludes the smell B (the confidence for the relation is lower than assumed limit). Knowing that, we may restrict the exploration area and limit the computational complexity of the smell detection process.

For example, a *Lazy Class*, which has no or limited functionality, is a negation of an over-functional *Large Class*. *Lazy Classes* are relatively easy to identify, since there exist numerous measures of functionality. Since we are aware of this smell presence, there is no need to look for *Large Class* symptoms. The latter smell embraces multiple subtle aspects, which are much harder to detect, so the knowledge of the *Lazy Class* presence allows giving up further exploration.

2.6. Other Relations

Remaining relations that may exist among the smells are insignificant for smell detection. Therefore we ignore them in further investigation.

3. Large Class Dependencies

Large Class is one of the most prevalent smells appearing in the code [4]. In this section we describe exemplary relations of simple compatibility, incompatibility, and aggregate compatibility affecting this smell, since it originates from various code flaws and is related to many other smells.

3.1. Simple Compatibility

Analyzing *Data Class* and *Feature Envy* smells definitions [4] we noticed that structure equivalent version of the former one is closely related to *Feature Envy*. If there is a structure equivalent class, there also must exist another one that uses its data. This class almost certainly contains methods that are *Feature Envy* candidates. Below there are the definitions of smells we used.

3.1.1. Data Class

A *Data Class* is a class inappropriately used as a data container [4]. There are three violations falling under this umbrella:

- Classes with public fields,
- Classes with improper collection encapsulation,
- Classes that have only getting and settings methods (structure equivalents); below there is an exemplary structure equivalent taken from Tomcat's [6] (*org.apache.catalina.deploy.FilterMap* class):

```
public class FilterMap implements Serializable {
    ...
    private String filterName = null;
    public String getFilterName() {
        return (this.filterName);
    }
    public void setFilterName(String filterName) {
        this.filterName = filterName;
    }

    private String servletName = null;
    public String getServletName() {
        return (this.servletName);
    }
    public void setServletName(String servletName) {
        this.servletName = servletName;
    }
    ...
}
```

Improper encapsulation allows a client of the class to modify its internals without notifying their owner, which may cause errors in program behavior [9].

A class that does not define its own methods also hinders code modifications [4], [9]. If representation of data within the class changes, then classes using the modified class will have to change, too. Therefore, it is recommended to aggregate methods that operate on data with the data itself.

There is also another aspect concerning *Data Class* smell. It is common that clients operate on data in the same way causing duplicated code smell to occur [9]. It is also worth to notice that a *Data Class* does not pay off for itself, because it deserves maintenance and documenting without giving any behavior in return [4].

In this paper we only detect the structure equivalent violations, because other were not related to the *Feature Envy* smell.

3.1.2. Feature Envy

A method that is more interested in a class other than the one it actually belongs to, is an example of *Feature Envy* smell [4]. Such a method should be moved to the class that it references the most. The code below is an exemplary *Feature Envy* method taken from Tomcat's *org.apache.catalina.core.ApplicationFilterFactory* class [6]:

```
private boolean matchFiltersServlet(
    FilterMap filterMap, String servletName) {
    if (servletName == null) {
        return (false);
    } else {
        if (servletName.equals(filterMap.getServletName())) {
            return (true);
        } else {
            return false;
        }
    }
}
```

This method checks if the servlet name given as the second method argument matches the filter's servlet name. The filter is given as the first method argument and this method does not use any of the fields and methods of its own class. As a result, this method should be moved to the *FilterMap* class.

Of course there are several design patterns, like *Strategy* and *Visitor* [10] that break this rule. In this article we did not take these cases under consideration.

3.2. Incompatibility

In order to examine this relation we examined the relation between *Data Classes* and *Inappropriate Intimacy* classes. The latter is defined as classes that "spend too much time delving in each other private parts" [4]. There are two violations covered by this smell:

- Bi-directional associations between classes, and
- Subclasses knowing more about their parents than their parents would like them to know.

Only the first one is covered in our evaluation, due to difficulties with automatic detection of the latter one.

The code below illustrates *Inappropriate Intimacy* smell. The following classes have bi-directional association (*org.apache.catalina.core.StandardContextValve* and *org.apache.catalina.core.StandardContext*):

```

final class StandardContextValve extends ValveBase {
    ...
    private StandardContext context = null;
    ...
    public final void invoke(Request request,
        Response response) throws IOException,
        ServletException{
        ...
        while (context.getPaused())
        ...
    }
}

public class StandardContext extends ContainerBase
implements Context, Serializable, NotificationEmitter {
    public StandardContext() {
        ...
        pipeline.setBasic(new StandardContextValve());
        ...
    }
    ...
}

```

StandardContextValve class invokes getPaused method in StandardContext class. StandardContext class invokes StandardContextValve constructor.

We identified *Data Classes* to be classes that are simple data holders and thus do not have bi-directional associations with other classes. In other words, if a class is *Inappropriately Intimate*, then it cannot simultaneously be a *Data Class*.

3.3. Aggregate Compatibility

The next bad smell taken into consideration was Large Class. According to Fowler [4] it is a class that has too much functionality. Over-functionality may result from improper class abstraction: several classes sum up together. Other reasons include the presence of Feature Envious methods or because the class is *Inappropriately Intimate* with other classes. Such a class needs to be split into smaller classes.

3.4. Transitive Compatibility

We have also discovered a transitive compatibility relation. The *Data Class* smell suggests the presence of the *Large Class* smell, because *Data Class* is related to *Feature Envy* and the *Feature Envy* is related to a *Large Class* smell.

4. Experimental Evaluation

The proposed approach to revealing compatibility and incompatibility relations was evaluated on Jakarta Tomcat 5.5.4 [6] source code. Tomcat is well known to the open-source community for its good code quality resulting from constant refactoring [11].

In subsequent sections we examine the presence of selected smell relations and evaluate their existence within Tomcat codebase.

4.1. Simple Compatibility

In order to evaluate a hypothesis that a *Data Class* suggests presence of the *Feature Envy*, we have adopted following definitions of these smells:

- A class is considered a *Data Class* if 80% or more of its methods are accessors/modifiers (getters and setters), and
- A method is considered *Feature Envious* if it references other classes more frequently than its own class's methods.

We analyzed only structure equivalent aspect of *Data Class*, because we found the other symptoms (improper encapsulation of fields and collections) are irrelevant for this evaluation.

With regard to *Feature Envy* we also inspected cases with equal number of external and internal references, although it is not considered a smell, because there is no need for moving a method (but it may be moved if it is also a *Data Class*).

Table 1 shows the results of the analysis of Tomcat code. We found 26 classes, which have at least 80% of accessor/modifier methods, and thus identified as *Data Classes*.

Table 1. Results of *Data Class* and *Feature Envy* smell relation analysis

Metric	Value
Total number of <i>Data Classes</i>	26
<i>Data Classes</i> referenced in <i>Feature Envy</i> methods	24
Number of <i>Data Classes</i> which were targets of <i>Move Method</i> refactoring	21
Number of remaining <i>Data Classes</i> after <i>Move Method</i> refactorings	7
Number of <i>Feature Envy</i> methods referencing <i>Data Class</i> classes	173
Number of <i>Feature Envy</i> methods mostly referencing <i>Data Classes</i> (a <i>Data Class</i> is the target of <i>Move Method</i> refactoring)	134
Number of <i>Feature Envy</i> methods referencing <i>Data Classes</i> equally frequent as its own class	43

Out of these, 24 representatives of *Data Class* smell were referenced in *Feature Envy* methods. In turn, the *Feature Envious* methods referencing *Data Classes*, the 77% of cases (134 of 173) reference the classes equally or more frequently than their own, which means they are subject to the *Move Method* refactoring. 43 out of 134 cases (32%) have equal number of external and internal references, which are attractive as well from the *Feature Envy* smell perspective. Therefore, in order to minimize the *Data Class* smell occurrence in such cases, the *Data Class* targets were preferred. This approach targeted 21 *Data Classes* and minimized the number of the smelly classes from 26 to 7.

The confidence level for the simple compatibility of *Data Class* and *Feature Envy* smells within the analyzed source code is then 0.92, and it is supported by 491 cases out of 830 considered. It shows that the knowledge of *Data Class* presence

significantly leverages detection of *Feature Envious* methods, and also suggests the direction of the *Move Method* refactoring to counteract the *Data Class* smell.

4.2. Incompatibility

Secondly, we analyzed the impact of incompatibility relation connecting *Inappropriate Intimacy* and *Data Class*. *Inappropriately Intimate* classes look in each other’s internals, whereas a *Data Class* is unilaterally exploited by its peers. Effectively, these smells exclude each other. In order to detect *Inappropriate Intimacy* we looked for bi-directional associations between classes. Those with at least one such association were considered to be *Inappropriate Intimacy* representatives. The analysis revealed that 159 of 830 inspected classes met this requirement.

Knowing that the *Inappropriate Intimacy* excludes *Data Classes*, the number of possible checks for the latter smell was reduced by 19%.

4.3. Aggregate Compatibility

Next, we analyzed the aggregate compatibility relation on the example of *Large Class* smell. We introduced four metrics that describe class functionality. Their definitions and accepted thresholds are presented in Table 2.

Table 2. Metrics used for measuring functionality (source: [12])

Metric	Description	Max. accepted
Number of Methods	Number of methods in the class	20
Weighted Methods per Class	Sum of cyclomatic complexities of class methods	100
Response for Class	Number of methods + number of methods called by each of these methods (each method counted once)	100
Coupling between Objects	Number of classes referencing the given class	5

We assumed that a class is considered large when at least one metric value exceeds the accepted threshold. Furthermore, we also noticed that a class is large when it is also *Inappropriately Intimate* or it has at least one *Feature Envious* method. Table 3 shows the results of the evaluation.

Table 3. Results of *Large Class*, *Inappropriate Intimacy* and *Feature Envy* smell relations analysis

Metric	Value
Total number of analyzed classes	830
Number of classes containing <i>Feature Envious</i> methods	463
Number of <i>Inappropriately Intimate</i> classes	159
Number of <i>Large Classes</i> found without exploiting dependencies between smells	230
Number of <i>Large Classes</i> found exploiting dependencies between smells	501

Metrics-only-based detector found 230 of 830 classes having too much functionality. After it has been provided with inter-smell dependencies data, the number of detected classes raised to 501. That doubled the effectiveness of *Large Class* detection.

5. Conclusions

The bad code smells indicators usually vary for every smell, but in some cases the smells share common roots. The existence of smells becomes then a valuable indicator of other flaws. Whereas it infrequently plays a primary role in smell detection, it could be successfully exploited as an auxiliary source of smell-related data.

There are several relations that exist among smells that could be used for that purpose. The analysis of a *Large Class* smell shows that several other code odors affect it and facilitate its discovery. The experiment with detecting *Large Classes* in Tomcat codebase shows that the knowledge of them may substantially support finding over-functional classes. We found examples of several smell dependencies, including simple, aggregate and transitive compatibility, as well as incompatible smells. We also determined the confidence ratio for selected inter-smell relations in that code, which suggest significant correlation among participating smells and usability of this approach to smell detection.

Several activities benefited from the dependency analysis between bad smells. Knowledge of inter-smell relations doubled the effectiveness and improved efficiency of *Large Class* detection. Also the number of possible checks for the *Data Class* smell was reduced by 19%, because of the fact that *Inappropriate Intimacy* excludes *Data Classes* presence.

Dependency analysis also helped in discovery of a transitive compatibility relation: the existence of *Data Class* smell implies *Large Class* smell, which broadens the set of smell dependencies. It also appeared helpful at removing the smells. Experimental evaluation revealed that the number of *Data Classes* was reduced from 26 to 7 when the relation between *Data Class* and *Feature Envy* smell was applied.

Further research directions on the inter-smell relations include exploration of a smell dependency map, which provides a complete set of relations among the odors, and building a prototype of smell detector that utilizes the previously identified flaws in finding other ones.

Acknowledgements

This work has been financially supported by the State Committee for Scientific Research as a research grant 4 T11F 001 23 (years 2002-2005).

References

- [1] Pearse T., Oman P.: Maintainability Measurements on Industrial Source Code Maintenance Activities. In: *Proceedings of International Conference of Software Maintenance 1995*, Opio (France), pp.295-303
- [2] Basili V., Briand L., Melo W.: A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Trans on Soft. Eng.*, Vol. 22 (1996) No. 10, pp.751-761

- [3] Beck K.: Extreme Programming Explained. Embrace Change. *Addison-Wesley*, 2000.
- [4] Fowler M.: Refactoring. Improving Design of Existing Code. *Addison-Wesley*, 1999.
- [5] Walter B., Pietrzak B.: Multi-criteria Detection of Bad Smells in the Code. In: *Proceedings of 6th International Conference on Extreme Programming*, 2005, Lecture Notes in Computer Science (in printing).
- [6] The Apache Jakarta Project: Tomcat 5.5.4, <http://jakarta.apache.org/tomcat/index.html>, January 2005.
- [7] van Emden E., Moonen L.: Java Quality Assurance by Detecting Code Smells. In: *Proceedings of the 9th Working Conference on Reverse Engineering*. IEEE Computer Press, 2003.
- [8] Agrawal R., Imieliński A.: Mining Associations between Sets of Items in Massive Databases. In: *Proceedings of the ACM SIGMOD-93 International Conference on Management of Data*, Washington D.C, 1993, pp.207-216.
- [9] JUnit, <http://www.junit.org>, January 2005.
- [10] Chidamber S.R., Kemerer C.F.: A Metrics Suite from Object-Oriented Design. *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, 1994, 476-493.
- [11] Marinescu R., Using Object-oriented metrics for Automatic Design Flaws Detection in Large Scale Systems. ECOOP Workshop Reader 1998, *Lecture Notes In Computer Science*; Vol. 1543, pp.252.255.
- [12] Gamma E., Helm R., Johnson R., Vlissides J.: Design Patterns. Elements of Reusable Object-Oriented Software. *Addison-Wesley*, 1995.
- [13] Tomcat Defect Metric Report, http://www.reasoning.com/pdf/Tomcat_Metric_Report.pdf, visited in April 2005.
- [14] NASA Software Assurance Technology Center: SATC Historical Metrics Database, <http://satc.gsfc.nasa.gov/metrics/codemetrics/oo/java/index.html>, January 2005.

Modeling and Verification of Reactive Software Using LOTOS

Grzegorz ROGUS and Tomasz SZMUC

*AGH University of Science and Technology, Institute of Automatics,
Al. Mickiewicza 30, 30-059 Cracow, Poland
e-mails: tsz@agh.edu.pl, rogus@ia.agh.edu.pl*

Abstract: The paper describes a possibility of LOTOS language application for modelling and verification of reactive systems in specification and design stages of development process. The main idea lies in an examination of behavioural equivalences between two specifications related to different abstraction layers, or between an ideal fault-free model and system (or its part) specification in current development stage. A framework for using LOTOS and observational equivalence for verification of artifacts in the above mentioned development stages is proposed in the paper. The concept is examined on example presented in the final part.

Introduction

Preserving correctness of software artifacts is a desirable feature of development process for reactive systems. The reason is quite obvious, some properties as concurrency, distributed architecture, nondeterministic of environment, etc. make practically impossible sufficient verification by testing. Moreover, reactive programs are very often used in the areas where the correctness and reliability of are of great importance. Typical applications for traffic control, production control, and many others where incorrect behavior may lead to disasters. Therefore, a research towards finding a proper formal methodological framework for systematic the design of reactive systems is needed.

Use of formal methods can support practitioners in two main ways. Firstly, by virtue of their mathematical basis, formal specifications allow clear, precise and unambiguous descriptions of a system; moreover, descriptions can be written without reference to implementation issues. Secondly, an existence of formal semantics for given specification language makes possible a rigorous mathematical analysis i.e. verification of the specification. Such analysis can improve confidence in the correctness of the design and implementation. It may also lead to a better, deeper understanding of the system, especially in the cases where the analysis detects some (non-trivial) error in the specification.

In the paper, it is presented a possibility of using formal (LOTOS) specification in verification reactive systems. The specification is translated into a Labeled Transition System LTS (a graph), where states in the LOTOS specification are mapped into LTS nodes, and LOTOS actions into LTS transitions correspondingly. The obtained LTS must comprise all the possible executions of the development system, but on other hand must remain finite to be generated.

Since the system is specified and designed in LOTOS, behavioral equivalence can be used to examine the relationship between two specifications $S1$ and $S2$. Those specifications may be related to required behavior ($S1$) and consequently to the developed system ($S2$) or two chosen specifications on different stages of development process.

The paper concentrates on presentation how the equivalence concepts may be applied in the verification. The proposed method is illustrated using a benchmark like alternating bit protocol example. The verification process is carried out automatically using CADP [3] package.

1. LOTOS Language

There are many different formal methods supporting design and verification of reactive systems: Petri nets, temporal logics or process algebras. The one chosen in the approach belongs to the family of process algebras, and is ISO standardized formal specification language [1]. The main benefits advocating for choosing this language are given below.

- Precise and unambiguous description of complex systems. LOTOS can describe a system at various abstraction levels with well-defined syntax and semantics. Therefore, LOTOS specifications are implementation independent and may be applied systematically in consecutive development stages, breaking the verification process into several simpler phases.
- Supporting different specification styles. There are many predefined styles in using LOTOS for building specifications: monolithic, state-oriented, resource-oriented, and constraint-oriented [9]. Constraint-oriented style can capture the properties of a system from different views or aspects. Constraint-oriented style is therefore widely used, as it supports stepwise development of system specification.
- System validation and verification. LOTOS has a strong mathematical basis [7]. The constructed models can be rigorously analyzed, verified, and validated. We can prove the correctness of LOTOS specifications using several methods based on theorem-proving or model-checking.
- Rapid prototyping. LOTOS specification are executable, because its semantics is operational. Therefore, it is possible to examine the behaviors and functionalities of systems at also at early development stages.
- Software generation. LOTOS specifications can be refined and extended step by step, leading to the final stage, where source codes may be generated.
- High level software reuse. Firstly, existing LOTOS components can be composed, modified, or generalized to produce new specifications, by applying new constraints. It is possible to automate the reuse process. Secondly, it is much easier to find appropriate reuse components using formal languages, because the functionalities are precisely defined. Thirdly, it is possible to assure the consistency of components integration.

The reasons above listed make language LOTOS specially useful for specification of reactive systems. LOTOS as concentrating on behavioral aspects, describes a system

by defining temporal relations between externally observable events at the so-called event gates. The language is composed of two parts: a process part based on the two formalisms: Milner's Calculus of Communicating Systems (CCS) [7] and Hoare's Communicating Sequential Processes (CSP) [6], and the second data algebraic part based on abstract data type language (ACT ONE). These two aspects are complementary and independent: the process algebra is used for modeling dynamic behavior of systems, and ACT ONE is applied to define data structures and value expressions.

The concept of LOTOS application for specification may be shortly described in the following way. On the basis of a given textual requirements an abstract representation of system as LOTOS specification is defined. The specification describes the expected and allowed behavior of prospective system. In the next steps the requirements are refined in analysis and design phases by adding the information of architecture, dividing the system into several subcomponents, refining the individual components, etc. Systematic correctness verification is carried out by proving selected equivalences between developed models. The paper concentrates on behavioral aspects developed systems (using Basic LOTOS).

2. System Development and Verification

Systematic correctness verification is significantly facilitated by stepwise specification of developed system. The development can be described as a sequence of specifications:

$$S0 \xrightarrow{P1} S1 \xrightarrow{P2} S2 \xrightarrow{P3} \dots \xrightarrow{Pn} Sn$$

where $S0$ is the first specification and Sn the last one.

In other words $S0$ is the most abstract specification, describing what the system must do without saying anything about how these actions are to be performed, while Sn will be much more detail, possibly providing implementation information. In this case $S0$ is the first attempt to formalizing informal requirements, while Sn is the specification from which we may attempt to derive program code (depending on available techniques).

For such class of specifications, the two sorts of verification can be proposed:

- Formal comparisons (proving equivalences) of any two selected specifications.
- Proving properties of an individual specification.

In paper we focus on the first sort, leaving the second one to more specific (application oriented) considerations. The verification (described in the next section) lies in proving that one specification is equivalent (in some sense) to another.

3. LTS and Equivalence

As was mentioned in the Section 1, LOTOS is specification language, which allows the specification of system at different abstraction levels. The relationships between

different LOTOS specifications of a given system can be studied using equivalence notions [7]. One of the most important among several proposed ones is the observational equivalence. This relation is based on idea that behavior of a system is determined by the way it interacts with external observers. This point of view implies that only visible action are concerned in this relation.

The underlying LOTOS model is based on a concept of Labeled Transition System (LTS). LTS is a generalization of finite state machine that provides a convenient way for expressing step-by-step operational semantics of behavioral expression. LTS consists of transitions the sort $s_1 \xrightarrow{a} s_2$, denoting that state s_1 can evolve into s_2 by execution of action a . LTS play a very important role in the verification concept presented in this paper, because, in fact, behavioral equivalences are proved between labeled transition systems.

Definition 1. (Labeled Transition System) A Labeled Transition System (LTS) is a tuple (S, Act, T, s_0) , where:

- S is a (finite) non-empty set of states;
- $s_0 \in S$ is an initial state;
- Act is a (finite) set of observable actions;
- $T = \{ \xrightarrow{a} \subseteq S \times S : a \in \text{Act} \cup \{i\} \}$ is a set of transitions describing by a binary relation on S ■

LOTOS behavior expressions frequently are shown as a behavior tree, which represents unfolded LTS (where loops are expended).

Bisimulation, simulation equivalences and preorders play a central role in the verification of communicating systems [4]. There are many efficient algorithms for computing various bisimulation equivalences (strong, weak, branching) [2]. Intuitively, two states p and q are bisimilar if for each state p' reachable from p by execution of an action a there exists a state q' reachable from q by execution of an action a , such that p' and q' are bisimilar. Formal definition is given below.

Definition 2. A relation $R \subseteq S \times S$ is a strong bisimulation if the following condition is satisfied:

If $(P, Q) \in R$, then $\forall a \in A$:

whenever $P' : P \xrightarrow{a} P'$, then $Q' : Q \xrightarrow{a} Q'$ and $(P', Q') \in R$

whenever $Q' : Q \xrightarrow{a} Q'$, then $P' : P \xrightarrow{a} P'$ and $(P', Q') \in R$ ■

Two LTS Sys_1 and Sys_2 are related modulo strong bisimulation denoted by $\text{Sys}_1 \sim \text{Sys}_2$ if it exists a strong bisimulation relation $R \subseteq S \times S$ between them.

3.1. Selection Relation in Verification Process

An abstract verification criterion may be chosen on the basis of a set of equivalence notions. Several simulation and bisimulation relations can be defined, such as strong simulation, strong bisimulation, branching bisimulation, safety preorder and safety equivalence [5].

Depending on the considered level of details (observable or hidden actions, branching structure, non-determinism) two specifications (LTS) may or may not be equivalent.

The crucial question here is which relation should be selected in the verification process. In the case of LOTOS one can choose three kinds of relations processes comparison: equivalence, congruence or preorder relations. The selection depends on some criteria and sort of development process (reduction, extension or refinement). If the development process changes alphabet of observable action, then selection of preorder relation is recommended. In another case equivalence relation may be selected. In most verification proofs, it is aimed at preservation of the substitutive property, i.e. that two equivalent processes will have the same behavior in all contexts (congruence relation). The congruence relation will be always chosen in preference to the corresponding equivalence relation. This recommendation applies also to weak bisimulation, congruence and testing congruence, strong equivalence, branching bisimulation equivalence and other congruence relations. The criterion also applies to preorder relations.

A selection of main sort of relation (for example equivalence) implies a choice of a kind of relation. The selection depends on properties which have to be proven. If i is viewed in the same way as all the other actions, then strong bisimulation is recommended. If i is treated as an unobservable action, then it cannot be used to distinguish between processes. In this case observation equivalence is preferred. Finally, when one is interesting in LTS structure, the branching equivalence should be selected. This relation also preserves safety and liveness properties.

4. Verification Behavioral Specification

Aldebaran (part of CADP [3]) is a tool for verification of communicating processes represented by LTS. LTS graph contains all possible execution sequences of constructed system. The verification process is shown in Figure 1.

The verification process consists of four steps:

1. Generation of LTS model using LOTOS specification and CAESAR (LOTOS compiler). The resulted model is represented as G_S .
2. Generation of LTS model using LOTOS specification and CAESAR. The resulted model is represented as G_I .
3. Initial reduction of processes G_S and G_I using for example strong bisimulation relation (G_{S-BS} , G_{I-BS}). This relation keeps all LTS properties, decreasing their sizes.
4. Comparison of G_{S-BS} and G_{I-BS} using Aldebaran (w.r.t. the chosen relation). When negative result is obtained, then it is generated initial sequence of actions, which don't realize proof relation.

The first step consists in application of CAESAR tool for generation of LTS graph from LOTOS specification. It is very important to generate a finite-state LTS model of

reasonable size. It is a reason why making some simplifications in constructed model is needed.

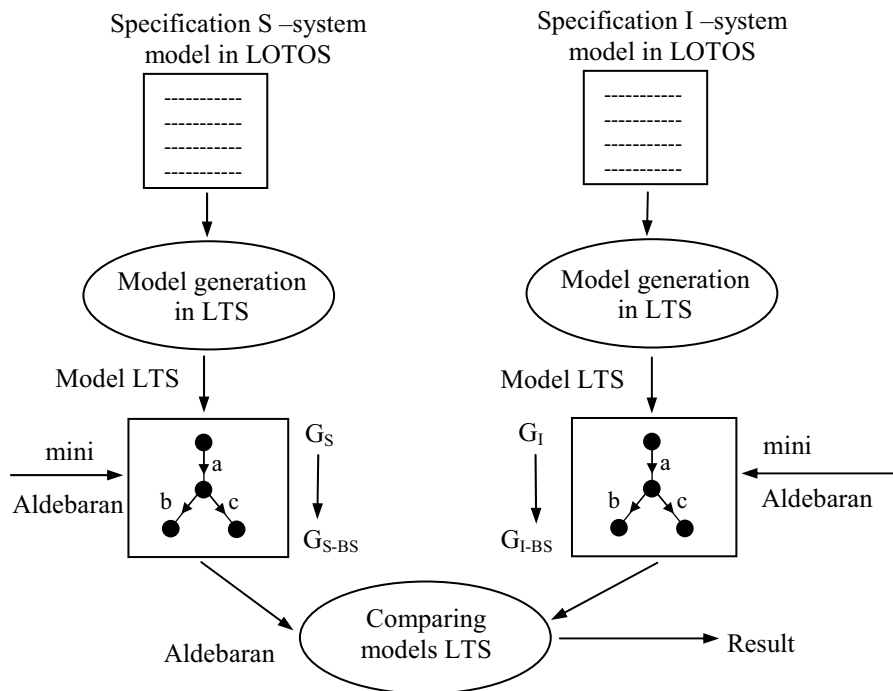


Figure 1. Verification by comparing models (using equivalence relation)

The second step lies in using Aldebaran tool for minimization of the resulted graph. A choice of relation depends on the demand which properties should be preserved. For example if one is interested in preserving safety properties, then the minimization may be carried out modulo safety equivalence. If LTS structure should be preserved, then branching equivalences is applied. Strong bisimulation is often used for initial model reduction. It is due to the fact, that this relation is neutral for model.

In the case when given models can not be verified, Aldebaran generates a diagnostic sequence. This sequence provides usually some help, however it only refers to a few non hidden actions that have been kept for their relevance to express the properties. It is called abstract diagnostic sequence. The diagnostic information may be enriched by getting detailed sequence (i.e. with more visible actions), that can clearly identify the scenario which goes to errors. This abstract diagnostic sequence should be encoded in the format suitable for input to the Exhibitor tool, which is then instructed to find the detailed sequence (allowed by the specification) matching the abstract one.

5. An Example – Alternating Bit Protocol

The proposed approach presented is illustrated using well known benchmark example – alternating bit protocol [8]. Process *Sender* sends messages to *Receiver* and each message sent over *data channel* is acknowledged by *Receiver*. The messages being sent are numbered modulo 2. If k -th k ($k \in \{0,1\}$) message *msg* is properly received, *Receiver* sends k -th ACK-signal *sendack* over the ACK channel.

In the first step two LOTOS specifications are created. The first specification *S1* is related to requirements analysis. Figure 2 represents specification *S1*, that consists of two process representing identify requirements: Sending and Receiving.

```
Specification ABP_Spec1 [get,give] :noexit
  Behavior
  (  hide sendmsg0,sendmsg1,sendack0,sendack1 in
    Sending [get, sendmsg0, sendmsg1, sendack0, sendack1]
    |[sendmsg0, sendmsg1, sendack0, sendack1]|
    Receiving [give,sendmsg0,sendmsg1,sendack0,sendack1]
  )
```

Figure 2. Specification of *S1* (requirements analysis)

Specification *S1* is an ideal model which has been created using constraint-oriented style in which only observable interactions are presented. It is assumed that no messages are lost in that model. In this step the focus is on specification correct sequential actions describing each requirement. Model is composed as a conjunction of separate constrains (functional requirements). Temporal ordering relationship between them generates behavioral model's view.

When sender plans to send message *get*, it passes the message to the receiver along with alternation bit 0 – *sendmsg0*. Message *get* is sent next, when sender receives an acknowledgement with the alternation bit set to 0 – *sendack0*. After this event sender takes next message *get*, adds alternation bit set to 1, and sends *sendmsg1* to receiver. When receive the acknowledgement with bit 1 *sendack1* all process restart. This is scenario for sender. Receiver works similarly, it waits for message *sendmsg0* or *sendmsg1*, and when gets message, then passes it to next layer of system – *give* and sends the acknowledgement with the correct alternation bit (*sendack0* or *sendack1*).

LTS graphs for specification *S1* are presented in Figure 3. These graphs have been created and then reduced using Aldebaran (simplification by: strong bisimulation, observation equivalence, safety equivalence and branching equivalence).

Preserving temporal ordering of actions *get* and *give* is the main requirement in this protocol. LTS from Figure 3 confirm this property. The first action is always message *get*, after which message *give* appears. It means that is impossible to receive message that was not send (sequential *give* – *get*) or sent a few messages after without acknowledgement on their reception (*get* – *get* – *get*). These properties are complied, hence the model *S1* is correct.

Specification for design phase development process is created in the next step. In the described example there are created three specifications: *S2*, *S3* and *S4*. Information on system's architecture and interface between components is introduced into the

developed model. The model for design phase has been created using mixed style of specifications, i.e. structure has been defined in the resource-oriented style, and behavior of all components has been specified in the state-oriented way.

In this stage the three components are considered: *Sender*, *Receiver* and *Channel*. *Chanel* consists of two subcomponents – channel *Trans* for sending messages with alternating bit, and channel *Ack* for sending acknowledgement.

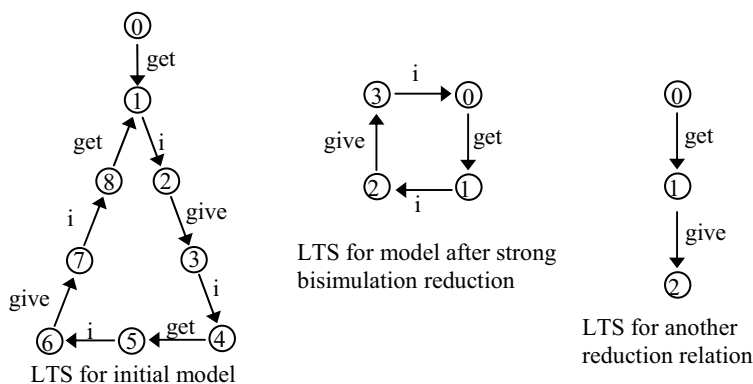


Figure 3. Graphs LTS to specification S1

The first specification S2 relates to ideal protocol, i.e. reliability of channels has been assumed. It means that the system guarantees temporal ordering of messages (communication channels as FIFO queues), and each message will be received (no messages lost).

Specification S2 is presented in Figure 4. Only main architecture level and a specification of one component have been shown.

```

Specification ABP_Spec2[get,give] :noexit
  Behavior
  ( hide sendmsg0,sendmsg1,sendack0,sendack1,
    recvmmsg0,recvmmsg1,recvack0,recvack1 in
    Sender [get, sendmsg0, sendmsg1, recvack0, recvack1]
    |[ sendmsg0, sendmsg1, recvack0, recvack1]|
    Channels [sendmsg0, sendmsg1, recvack0, recvack1,
              recvmmsg0, recvmmsg1,sendack0, sendack1]
    |[ recvmmsg0, recvmmsg1, sendack0, sendack1]|
    Receiver [give, recvmmsg0, recvmmsg1, sendack0, sendack1]
  )
  where
  Process Sender[get,sendmsg0,recvack0,sendmsg1,
                 recvack1]:noexit:=
    ReadySend0[get,sendmsg0,recvack0,sendmsg1,recvack1]
  where
    process ReadySend0[get,sendmsg0,recvack0,sendmsg1,
                      recvack1]:noexit :=
      get;

```



```

    sendmsg0;
    WaitAck0[get, sendmsg0, recvack0, sendmsg1, recvack1]
endproc
process ReadySend1[get, sendmsg0, recvack0, sendmsg1,
    recvack1]:noexit :=
    get;
    sendmsg1;
    WaitAck1[get, sendmsg0, recvack0, sendmsg1, recvack1]
endproc
process WaitAck0[get, sendmsg0, recvack0, sendmsg1,
    recvack1]:noexit :=
    recvack0;
    ReadySend1[get, sendmsg0, recvack0, sendmsg1, recvack1]
endproc
process WaitAck1[get, sendmsg0, recvack0, sendmsg1,
    recvack1]:noexit :=
    recvack1;
    ReadySend0[get, sendmsg0, recvack0, sendmsg1, recvack1]
endproc
endproc

```

Figure 4. Specification S2 (design phase)

Some properties of LTS graphs generated by reduction S2 are shown in Table 1.

Table 1. Verification of models for specification S2

Kind of equivalence	States	transitions	Deadlock?	Livelock?
Specification S2	13	13	No	no
Strong bisimulation	6	6	No	nie
Observation equivalence	2	2	No	no
Branching equivalence	2	2	No	no
Safety equivalence	2	2	No	no

In the next step a comparison of two models *S1* and *S2* using different equivalence relation is performed. The models are checked after initial reduction by strong bisimulation. The results are shown in Table 2.

Table 2. Results of comparison *S1* and *S2* specifications

Relation	Result
Strong bisimulation	false
Observation equivalence	true

Branching equivalence	true
Safety equivalence	true

On the basis of the results it may be stated that S1 and S2 models are consistent. Both models satisfy branching and safety equivalences. It means, that S1 and S2 comply the same set of safety and liveness properties (without divergences in models). Therefore it may be concluded, that the development process is correct (both models complied the same requirements).

In the next specification S3 it is assumed, that communication channels may lost messages. These channels are defined as a buffer (1 message size). The corresponding process is shown in Figure 5.

```

process K_Trans[sendmsg0,rcvmsg0]:noexit :=
    hide lost1 in
    sendmsg0;
    (
        rcvmsg0;
        K_Trans [sendmsg0,rcvmsg0]
    []
        lost1;
        K_Trans [sendmsg0,rcvmsg0]
    )
endproc

```

Figure 5. Channel *Trans* in specification S3

Consequently, Table 3 summarizes some properties of LTS graphs generated by reduction of S3 specification.

Table 3. Models for specification S3

Kind of equivalence	States	transitions	Deadlock?	Livelock?
Specification S3	17	17	yes	no
Strong bisimulation	7	9	yes	no
Observation equivalence	5	6	yes	no
Branching equivalence	5	6	yes	no
Safety equivalence	2	2	no	no

Verification of S1 and S3 models leads to negative results. A comparison using safety equivalence relation generates only OK as a result (both models complied the same properties – temporal ordering messages *get*, *give*). The others comparisons lead to negative result. Additional problem is related to an appearance of deadlock in some

models. This is caused by the fact, that messages can be lost in channels and it is no resending message construct in the specification. This situation is shown in Figure 6.

Specification *S4* is proposed in order to avoid the described problems (*S3*). Functionality of component *Sender* is changed in the first step. Figure 7 presents state diagram for this component. A possibility of resending messages (*sendmsg0*, *sendmsg1*) in the case loosing message has been added.

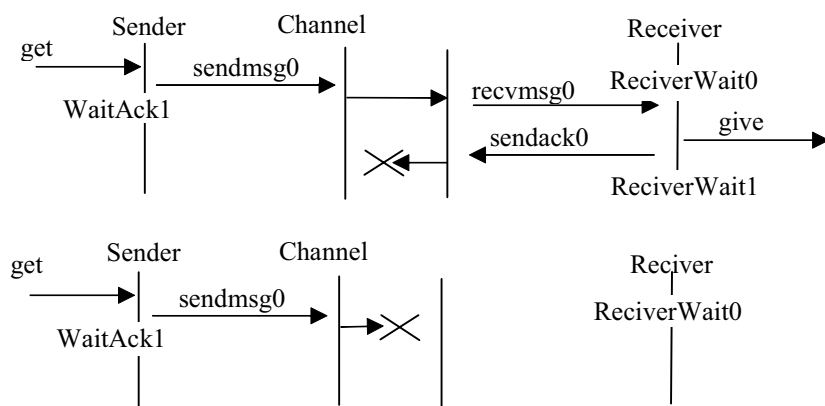


Figure 6. System's behavior when message is lost in *S3*

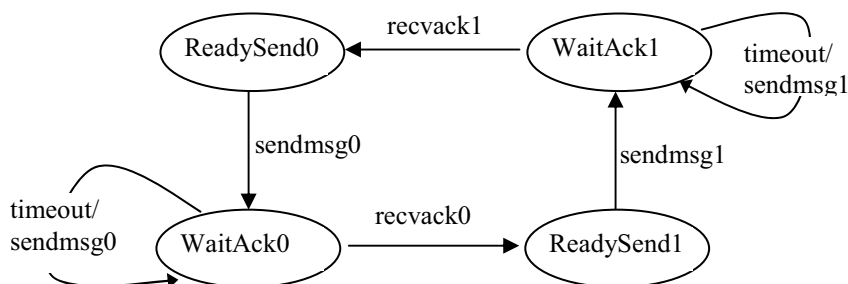


Figure 7. State diagram for component *Sender*

Table 4. Models for specification *S4*

Kind of equivalence	States	transitions	Deadlock?	Livelock?
Specification <i>S4</i>	257	603	no	no
Strong bisimulation	194	451	no	no
Observation equivalence	64	139	no	no
Branching equivalence	64	155	no	No
Safety equivalence	11	15	no	No

As it has been shown in Table 4, in specification *S4* deadlock has not appeared. In the next steps *S1* and *S4* models are compared. In all the cases negative result has been obtained. It means, that model *S4* is not correct in the sense of behavioral equivalence. Very important is fact, that safety equivalence is not preserved here. Detail analyses shows an interesting result, i.e. specification *S4* does not guarantee temporal ordering of actions *get* and *give*. This is a result of the introduced way of modeling of communication channels.

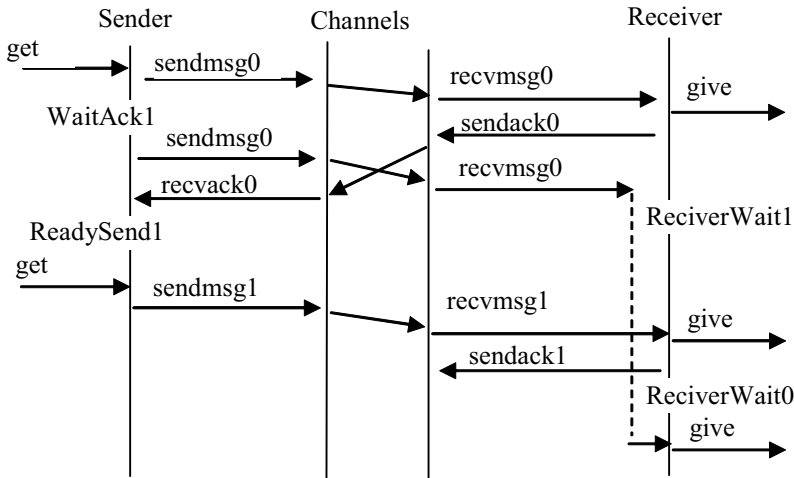


Figure 8. An example of incorrect behavioral specification *S4*

They do not work as FIFO queues, because in the model a connection to destination may be selected. This is a reason why the temporal order messages is not guaranteed. An example of incorrect behavior is shown in Figure 8.

Verification using equivalence for comparison of two models provides information on potential problems in specifications (implementation). In the next step simulation of the implementation (using Simulator from CADP package) has been performed, and provides positive results. It should be concluded, that an absence of equivalence relation between two models, is the first notice on correctness of development process.

6. Conclusions

The paper presents an approach to systematic development of correct reactive software using LOTOS language for formal specification. The specification language and related set of equivalences allow systematic correctness checking during consecutive steps of development process. CADP package is used for automatic verification of the related models. The paper concentrates on presentation of the main concept (Introduction and Sections 1-3) and relatively wide its illustration using a benchmark example (alternating bit protocol). Description of more advanced application of the approach may be found in [8], where a proposal of a methodology based on existing SDL

language and complementary LOTOS language presented and analyzed focusing on correctness verification. More detailed description of the approach and other examples of applications may be found there.

References

- [1] Bolognesi T. and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1), 1987.
- [2] Bolognesi T. and Smolka S.A. Fundamental results for the verification of observational equivalence. In H.Rudin and C.H. West (eds) *Protocol Specification, Testing and Verification VII*, 1987.
- [3] CADP, <http://www.inrialpes.fr/vasy/vadp.html>
- [4] Fernandez J. C. and Mounier L. On the fly verification of behavioral C. equivalences and preorders. In *CAV 91: Symposium on Computer Aided Verification*, Alborg, Denmark, June 1991.
- [5] Fernandez J-C., Mounier L. A Tool Set for deciding Behavioral Equivalences. <http://www.inrialpes.fr/vasy/Publications/cadp.html>, 1991.
- [6] Hoare C.A.R. Communicating sequential processes. *Prentice-Hall International*, 1985.
- [7] Milner R. A calculus of communicating systems. *Lecture Notes in Computer Science*, volume 92, Springer-Verlag, Berlin, 1980.
- [8] Rogus G.: Applications of LOTOS language to support development of correct software for reactive systems, Ph.D. Thesis (supervisor T. Szmuc), AGH University of Science and Technology, 2005
- [9] Vissers C., Scollo G., and M. Van Sinderen. Architecture and specification style in formal descriptions of distributed systems. *Protocol Specification, Testing, and Verification, VIII*, North-Holland, 1988.

This page intentionally left blank

8. Selected Topics in Software Engineering

This page intentionally left blank

Evaluation of Software Quality

Krzysztof SACHA
Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warszawa, Poland
e-mail: k.sacha@ia.pw.edu.pl

Abstract. The paper describes a method, which we used to evaluate the expected quality of software that was developed for a huge governmental system. The evaluation lasted nearly two years and was performed along with the software development process. The output that was expected by our customer consisted of a quality assessment accompanied by a set of recommendations on what to do in order to enhance the quality of the product.

Introduction

The ultimate goal of software engineering is to find methods for developing high quality software products at reasonable cost. As computers are being used in more and more critical areas of the industry, the quality of software becomes a key factor of business success and human safety.

Two approaches can be followed to ensure software quality. One is focused on a direct specification and evaluation of the quality of software product, while the other is focused on assuring high quality of the process by which the product is developed.

The software industry is currently entering a period of maturity, in which particular informal approaches are specified more precisely and are supported by the appropriate standards. Quality characteristics of software products are defined in ISO/IEC 9126 [1]. For each characteristic, a set of attributes which can be measured is determined. Such a definition helps in evaluating the quality of software, but gives no guidance on how to construct a high quality software product.

The requirements for a quality management system are defined in ISO 9001 [2]. All the requirements are intended for application within a software process in order to enhance the customer satisfaction, which is considered the primary measure of the software product quality. The quality management system, as defined by the standard, can be subject to a certification.

This paper describes a method, which we used to evaluate the expected as well as the actual quality of a huge software system that was developed in the years of 2003-2004 to support Common Agriculture Policy of European Union in Poland (IACS – Integrated Administration and Control System). Both of the two approaches, mentioned above, appeared to be too abstract for a direct application. Moreover, the quality of a software product cannot be evaluated unless the development of this particular product has been finished, and high quality of a software process does not necessarily guarantee high quality of the product. Therefore we were pushed to develop an original method. A general framework of this method has been described in [3].

The paper is organized as follows. Section 1 provides the reader with a short overview of the approach represented by ISO 9126, and Section 2 summarizes the approach of ISO 9001. The method that was used to evaluate the quality of the development of the IACS software is presented in Section 3. Final remarks are gathered in Conclusions.

1. ISO/IEC 9126 Overview

ISO 9126 [1] is concerned primarily with the definition of a quality model, which can be used to specify the required product quality, both for software development and software evaluation. The model consists of six quality characteristics, which are intended to be exhaustive. Each quality characteristic is very broad and therefore it is subdivided into a set of sub-characteristics or attributes. This quality model can be applied in industry through the use of measurement techniques and related metrics.

All the six quality characteristics, defined in ISO 9126, are recapitulated below, along with some comments.

Functionality is defined as the ability of the software product to provide functions which meet stated or implied needs of the user. This is a very basic characteristic, which is semantically close to the property of correctness, as defined in other quality models [4]. If software does not provide the required functionality, then it may be reliable, portable etc., but no one will use it.

Efficiency is a characteristic that captures the ability of a correct software product to provide appropriate performance in relation to the amount of resources used. Efficiency can be considered an indication of how well a system works, provided that the functionality requirements are met. The reference to the amount of resources used, which appears in this definition is important, as the traditional measures of efficiency, such as the response time and throughput, are in fact system-level attributes.

Usability is a measure of the effort needed to learn and use a software product for the purpose chosen. The scope of this factor includes also the ease of assessment whether the software is suitable for a given purpose and the range of tolerance to the user errors. The features that are important within the context of usability are adequate documentation and support, and the intuitive understandability of the user interface.

Reliability is defined as the ability of software to maintain a specified level of performance within the specified usage conditions. Such a definition is significantly broader than the usual requirement to retain functionality over a period of time, and emphasizes the fact that functionality is only one of the elements of software quality that should be preserved by a reliable software product.

Maintainability describes the ease with which the software product can be analyzed, changed and tested. The capability to avoid unexpected effects from modifications to the software is also within the scope of this characteristic. All types of modifications, i.e. corrections, improvements and adaptation to changes in requirements and in environment are covered by this characteristic.

Portability is a measure of the effort that is needed to move software to another computing platform. This characteristic becomes particularly important in case of an application that is developed to run in a distributed heterogeneous environment or on a high performance computing platform, which lifespan is usually short. It is less important if the application runs in a stable environment that is not likely to be changed.

It can be noticed that the characteristics correspond to the product only and avoid any statement related to the development process. This way the standard presents a coherent model, which skips such features like e.g. **timeliness**, which can be defined as the ability of a product to meet delivery deadlines. Although timeliness relates closer to the process than to the product itself, it can influence the subjective feeling of the customer. If a product is delivered late, then it may be of good quality, but customers may consider it to be of lesser quality than the products delivered on time [5].

2. ISO 9001 Overview

ISO 9001 [2] describes the requirements for a quality management system, which is a part of the total manufacturing process. The standard is very general and applies to all types of organizations, regardless of their size and of what they do. The recommended practices can help both product and service oriented organizations and, in particular, can be used within the context of software development and manufacturing. ISO 9001 certificates are recognized and respected throughout the world.

Because of this generality it is not easy to map the recommendations of the standard into the practical activities that can be performed within a software process. Moreover, the standard is intended to be used by the manufacturers and not by the auditors that are hired by their customers. Therefore it contains many recommendations that relate to resource management process, which was completely outside the scope of our evaluation. What we were expected to assess was the quality of products of particular steps of the development process: analytical specifications, design documents, testing plans and procedures, user manuals and the resulting code. The actual implementation of the testing process was also subject to our evaluation.

ISO 9001 does not define any particular model of quality. Instead, it adopts a simple approach that the quality of a product is measured by the customer satisfaction. According to this approach, no quality characteristics are defined, and the only basis for quality evaluation are the customer requirements. If those requirements are met, then the product quality can be evaluated high. The lack of a quality model makes this standard orthogonal to ISO 9126. There are no common points between the two, but also no contradiction can be found.

The top level requirement of ISO 9001 is such that a quality management system must be developed, implemented and maintained. All the processes and activities performed within the scope of this system have to be documented and recorded for the purpose of future review. A huge part of the standard relates to the processes of quality planning and management, resource management, and continuous quality monitoring, analysis and improvement. This part is not very helpful in evaluating the quality of a specific software product under design.

The part, which relates directly to the body of a software project, is a section on realization requirements. Basic requirements and recommendations that are stated therein can be summarized as follows:

1. Identify customers' product requirements, i.e. the requirements that the customer wants to meet, that are dictated by the product's use or by legal regulations.
2. Review the product requirements, maintain a record of the reviews, and control changes in the product requirements.

3. Develop the software process, clarify the responsibilities and authorities, define the inputs and outputs of particular stages.
4. Perform the necessary verification and validation activities, maintain a record of these activities, and manage design and development changes.

All of those statements are very concrete and provide valuable guidelines for auditing and evaluating the quality of a software process. Moreover, the stress that is placed on the need to meet customer requirements helps in closing the gap between the quality of the software process and the quality of software itself.

3. Quality Evaluation Method

There is a great difference between the quality evaluation made for and by a software manufacturer and the evaluation that is made for the customer.

A manufacturer can define a set of metrics that describe particular quality characteristics, measure and collect historical data from a set of similar projects, and compare the current data with the ones taken from the historical database. The entire process can be supported by computerized tools like the one described in [6]. Such an approach is focused on a direct evaluation of the quality of a software product, and can be implemented using GQM method (Goal – Question – Metric), described for the first time in [7] and developed since that time by NASA. The set of goals or quality characteristics can be the same or similar to the one defined in ISO/IEC 9126.

It is very difficult to a customer to follow this approach. The lack of historical data makes many of the quantitative characteristics useless. For example, consider the following metric: “The percentage of classes that contain a method for displaying help information for the user” (Table 1 in [6]) and assume that it has been measured equal to 45%. What does this data mean? Is 45% many or few? How does 45% contribute to the usability of the software product – is it high or low?

3.1. Process-Centric Approach

The problems related to the interpretation of many quantitative metrics of software quality pushed us towards the approach focused on the evaluation of the quality of the development process. The rationale that stands behind this approach is based on a hope that if things are done right, then the results will also be right. Since *hope* is not the same as *certainty*, we tried to relate somehow the attributes of process quality to the attributes of product quality defined in ISO 9126. An important advantage of such a process-centric approach is that it gives the auditor an opportunity to formulate recommendations on what to improve in the development process.

The method we developed was based on a set of criteria that characterize the quality of the software process or the quality of a product of a particular step of the software process. The method was similar to GQM in that the evaluation of particular criteria was based on a process of asking questions and giving answers to these questions. To avoid problems with the interpretation of data, the sets of questions were limited to the ones that were meaningful for the user. To make the evaluation process structured, we divided the problem space into six subject areas. The first subject area corresponded to the development process itself, the next four areas corresponded to

particular activities within the process, and the last one dealt with the software documentation. This way the following areas were defined:

1. Software process and development methods.
2. The analysis and analysis products.
3. The design and design products.
4. The implementation and the code.
5. Testing process and test documentation.
6. User manuals.

It can be noted from the above list that the areas 2 – 5 cover all the activities that exist in both: waterfall and incremental models of software development. The decomposition of the software process into the subject areas is then exhaustive.

Each subject area was covered by a set of criteria, organized along the paths of traceability: from requirements to verification, from requirements to the design, and from the design to the implementation. The majority of answers was qualitative. Quantitative measures were also used, however, only in such cases in which the numbers could be meaningful for the customer. Sample criteria and questions that were stated in particular subject areas, are discussed briefly in the next two subsections. The mechanics of the quality evaluation is described in Section 3.4.

3.2. Software Process and Development Methods

The main goals of the activities performed within this subject area are the identification of methods and standards that were used throughout the development process and the verification of the use of those methods with respect to completeness, readability and traceability of the resulting products. In order to achieve these goals we defined the following set of criteria, accompanied by the appropriate questions:

1. Methods and standards: Which methods and standards were used throughout the development process? How was the scope of the methods?
2. Completeness of results: Which artifacts recommended by the methods have been created? Is the set of artifacts sufficient? Are the results documented properly?
3. Readability and modifiability of the documentation: Are the created documents readable and modifiable?
4. Traceability of the documentation: Are the documents created in subsequent steps of the development process traceable?

As can be seen from the above list of criteria, the evaluation within this subject area was focused on formal aspects of the development in that it did not include an in depth analysis of the contents of the documents created.

The criteria correspond to the recommendations of ISO 9001, which state that a software process shall be developed, the outputs of particular activities shall be defined, and the results shall be recorded. The criteria have also a clear relation to the quality characteristics defined in ISO 9126, because the completeness of results together with readability, modifiability and traceability of the documentation promotes the maintainability and portability of the software product.

3.3. Analysis

The main goals of the activities performed within this subject area are the identification of methods and models that were used throughout the analysis and the verification of the use of those methods with respect to correctness, completeness and verifiability of the resulting documents and other products. In order to achieve these goals we defined the following set of criteria, accompanied by the appropriate questions:

1. Completeness of sources: Which sources of information were used throughout the analysis? Was the selection of legal regulations complete?
2. Consistency of the analysis model: Is the business model created during the analysis consistent with legal regulations that have been identified in criterion 1?
3. Completeness of the analysis model: Is the created analysis model complete in that it comprises all the business procedures defined by the legal regulations? Is the list of possible scenarios complete?
4. Completeness of the context definition: Is the set of input data sufficient to achieve the business goals of the system? Is the set of output data complete with respect to business and legal requirements?
5. Completeness of the functionality: Are the functional requirements complete in that all the business procedures identified in criterion 3 are supported by the appropriate functions of the software?
6. Usability of the user interface prototype: Is the prototype complete and ergonomic?
7. Correctness of the data model: Is the data model complete and consistent? Is the data model consistent with the business procedures? Does the data model comply with the best engineering practices?
8. Completeness of the non-functional requirements: Are the non-functional requirements complete in that they define the expectations related to the security of data, response time, throughput and reliability?
9. Verifiability of the non-functional requirements: Are the non-functional requirements defined verifiable (testable)?
10. Credibility of the verification: How is the coverage of business procedures by the test scenarios? Is the performance testing sufficient? How is the credibility of the testing procedures?

As can be seen from the above list of criteria and questions, the evaluation within this subject area is focused on the contents of the analytical products. The sequence of questions moves along a path: From sources of information to business model, from business model to functions and efficiency, from functions and efficiency to verification. The results of the evaluation are in good relation to the following characteristics of ISO 9126: Functionality is supported by criteria 2, 3, 4, 5, 7, efficiency by criteria 8, 9, usability by the criterion 6, reliability by criteria 8, 9 10, and maintainability by the criterion 7. Portability was not considered a significant premise at the analysis level of IACS.

3.4. Evaluation Process

Input data to the quality evaluation process consisted of all the documents that had been created in the entire software development cycle. These included:

- a business model developed in the inception phase,
- early analysis model of the elaboration phase,
- the set of analysis and design models created in the construction phase,
- the code and the complete set manuals,
- test plans and test reports (plus the observation of the testing process).

Because of the incremental nature of the development, part of the documents circulated in several versions, issued in a sequence of subsequent increments.

The evaluation process was decomposed into the areas listed in Section 3.1 and structured according to the set of criteria exemplified in Sections 3.2 and 3.3. The analysis that was done within the context of a particular criterion was guided by the set of questions. Sometimes, a compound area was decomposed into sub-areas and the questions were stated and answered separately for particular groups of artifacts. For example, the evaluation of the criterion 1 of Section 3.2 (Methods and standards) was decomposed into a review of the software process, analysis methods, design methods, implementation methods and tools, testing methods, and documentation standards.

The answers to the questions were, in general qualitative, formulated usually in terms of a ranking: good, satisfactory, bad. Quantitative metrics were also used, however, only in such cases in which the results could be meaningful to the user. An example of such a metric is the coverage of use case scenarios by test scenarios.

The evaluation report was structured in accordance with the areas and the criteria. The results of the evaluation within a particular area were concluded in the form of two sections: Risks for the project and recommendations to the project. The risk section summed up the 'bad' answers and related the deficiencies to the quality characteristics of ISO 9126. For example:

- The lack of functions that support certain business procedures creates the risk that the functionality of software will not be met.
- The lack of readable design models creates the risk that the maintainability of software will be low.

The recommendation section advised what to do in order to avoid the risks identified in the evaluation process and described in the previous section. An advice related to the first point above could, for example, be: Define the functionality that is missing.

4. Conclusions

This paper describes a practical method that can be used to evaluate the expected quality of software under design. The evaluation process does not refer directly to the existing standards, however, it is consistent with the basic elements of the quality models of both ISO 9001 and ISO 9126. The mechanics of the evaluation is based on

a set of criteria that are decided by stating questions and finding answers to those questions. The collection of criteria is structured into a set of subject areas that exhaustively cover the set of activities that exist in the most popular software processes: Phases of the waterfall model or activities of the RUP incremental process.

The method was used successfully in evaluating the quality of software products during the development of IACS system. The evaluation was performed on behalf of the customer and not the manufacturer of the system. The criteria and questions that guided the evaluation process allowed for a systematic, in depth analysis of the deliverables of the particular development activities. As result, several risks were revealed and identified. The recommendations helped in avoiding the risks in the final product. IACS system was build and certified for use within the deadline.

The advantages of the method can be summarized as follows:

- The method does not depend on any particular software process or method that can be used in the development of software.
- The results of the method, i.e. the answers to the questions that are stated in the evaluation process, are readable and meaningful to the customer.
- Negative answers to particular questions can easily be translated into recommendations on what to change in order to enhance the quality of products.

The method is simple in use, does not relay on any historical data, and need not be supported by a computerized tool.

References

- [1] ISO/IEC 9126-1, Software engineering – Product quality. ISO/IEC (2001)
- [2] ISO 9001, Quality management systems – Requirements. ISO (2001)
- [3] Zalewski A., Cegiela R., Sacha K.: Modele i praktyka audytu informatycznego, in: Huzar Z., Mazur Z. (eds.): Problemy i metody inżynierii oprogramowania, *WNT*, Warszawa (2003)
- [4] Fenton, N.: Software Metrics: A Rigorous Approach, *Chapman and Hall* (1993)
- [5] Horgan G., Khaddaj, S., Forte P.: An Essential Views Model for Software Quality Assurance, ESCOM-SCOPE 99 (1999)
- [6] Szejko S.: RDQC – sterowana wymaganiami kontrola jakości oprogramowania, in: Górski J., Wardziński A. (eds), Inżynieria oprogramowania: Nowe wyzwania, *WNT*, Warszawa (2004)
- [7] Basili V.R., Weiss D.M., A Methodology for Collecting Valid Software Engineering Data, *IEEE Transactions on Software Engineering*, Nov. (1984)

CMMI Process Improvement Project in ComputerLand

Lilianna WIERZCHON
*ComputerLand S.A.,
Organization and Management Department,
Al. Jerozolimskie 180, 02-486 Warsaw, Poland
e-mail: lwierzchon@computerland.pl*

Abstract. The paper describes the approach, which was applied by Polish IT company ComputerLand in CMMI Process Improvement Project. The free tool Interim Maturity Evaluation based on Capability Maturity Model Integrated for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing, v1.1 was applied to introduce to ComputerLand's staff the CMMI model and perform self assessment of the current processes in development, implementation and project management teams.

1. CMMI Process Improvement Project in ComputerLand

The main objectives of CMMI [1] Project in ComputerLand (CL) are: appraise organizational maturity, establish priorities for improvements and implement these improvements according to CMMI model.

In 1996 CL obtained an ISO 9001 certificate in the area of designing, building and integration of computer networks including necessary hardware and in servicing. In 2001 the certificate was widened to include designing, production, customization, testing, implementation and servicing of in-house developed software and CL's partners application software according to ISO 9001:2000 standard [5], [6], [7].

Therefore, based on the comparison of these two models [2], [3], CL is expected to be qualified between level 2 and level 3 of CMMI. To achieve CMMI 3 level CL started within the CMMI Project five corporate subprojects to cover known gaps and improve current organizational solutions:

- Central HelpDesk Project to integrate Central Hardware HelpDesk and local Software Product Helpdesks,
- Knowledge Management Project to integrate local knowledge databases and project repositories,
- Testing Project to establish separate System Testing Department,
- Estimation Project to establish and implement the standards and procedures for estimating,
- Central Project Management Project to choose and implement a uniform, coherent, integrated tool for project management and reporting.

For CMMI Project CL applies the following approach:

- Disciplines: SE, SW, IPPD, SS,
- Representation: Staged,
- Scope of Project: Development and Implementation Department (DID), Project Management Department (PMD),
- Internal training: internal CMMI and assessment methods training for key CL staff (e.g. Managers, Project Managers, Analysts, Designers, Developers, Testers, Quality Staff),
- Self assessment: self assessment for several instances of processes at 2nd and 3rd maturity level in DID and PMD
- Improvement: realization of 5 corporate subprojects and implementation of corrective and improvement activities to cover the gaps found during self evaluations,
- Consulting company: choice of CMMI consulting company with Certified Trainer and Certified Lead Appraiser,
- External training for 15 % – 40 % of CL staff (E.g. Managers, Project Managers, Analysts, Designers, Developers, Testers, Quality Staff): Introduction to CMMI, Introduction to SCAMPI Method,
- External appraisal: 3-level SCAMPI (Standard CMMI Appraisal Method for Process Improvement): Standards and Procedures Conformance Review (Class C appraisal), Gap Analysis (Class B appraisal), Final appraisal (Class A appraisal),
- Improvement: implementation of corrective and improvement activities to cover the gaps found during Standards and Procedures Conformance Review and Gap Analysis.

2. Self Assessment Methodology

For self assessment we used the free tool Interim Maturity Evaluation (IME) for CMMI SE/SW/IPD/SS stage representation, designed by Marc De Smet from Management Information Systems [4]. IME helps to track progress in Process Improvement (PI) projects, provides an organization with a means for self evaluation (it is recommended that an external CMMI consultant guides the IME process the first (few) time(s)), makes it possible to involve any person of an organization in PI activities, thus increasing their awareness, understanding and involvement, is a means to further educate people in an organization in the CMMI.

IME consists of:

- Presentation with a simple description of IME and CMMI,
- Set of questionnaires, which contains tables with numbered specific and generic practices and place for score by maturity level and process area,
- Spreadsheet with sheets compliant with set of questionnaires, in which it is possible to trace ranking of each process area and maturity level.

According to IME moderator organizes and performs assessment meeting.

Before the assessment meeting the moderator:

- selects a number of participants (1 to 30, typically 4-6),
- sends invitations to the participants who represent different roles.

During the meeting:

- moderator gives introduction (defining CMMI terms needing interpretation and explaining IME and the scoring),
- moderator hands out IME questionnaires to participants,
- per Process Area questionnaire participants fill in the questionnaire – they score a number from 0 to 10:

0-1	this practice is not required and is (almost) never done
2-3	this practice is sometimes required or is sometimes done
4-5	this practice is required but not always done, or the practice is regularly performed although is not required or not checked
6-7	this practice is normally required and usually done
8-9	this practice is required, is done and it is checked (the practice is institutionalized)
10	this practice is institutionalized and is a world class example,
“?”	for ‘I don’t know’
“na”	for ‘this is not applicable’, thus provide additional information,
- moderator enters the results in the spreadsheet,
- large deviations in opinions are discussed (possibly leading to corrections),
- spreadsheet shows the results of the last 6 IMEs and allows comparison between the data.

After the meeting the moderator:

- generates the IME report: average score per practice within process area, including number of “?” and “na” and radar diagram for each maturity level with the average score per process area,
- adds recommendations for process improvement,
- distributes the IME report and recommendations.

3. Self Assessment Conducting

CL Organization and Management Department (OMD) organized in March-July 2004 some internal trainings on CMMI, SCAMPI and IME for key managers and over 60 engineers from DID and PMD. Higher management chose projects and products for IME evaluation. Afterwards the self assessment was preformed with on-going support of OMD:

- in Project Management Centers of PMD for 18 projects realized for customers, where each project was treated as an instance of CMMI processes.
- in Competence Centers of DID for 28 software and hardware products, where management, development and maintenance of each product was treated as an instance of CMMI processes,

2-3 key engineers, a quality assurance engineer and a manager responsible for a project (project manager) or a product (product manager) as a moderator took part in

each evaluating meeting. Each meeting lasted 6-8 hours. After the meetings the moderators generated the IME report, added their recommendations for process improvement and distributed the IME report and recommendations to the meeting participants, managers responsible for competence centers or project management centers, directors of proper departments. IME reports and data were sent to OMD to generate individual (for each instance) and a summary (for all the company) SCAMPI reports and charts using an on-site developed tool which helped to analyze gaps and problems on the corporate level and to elaborate the corporate recommendations. Scoring for generic and specific goals and generic and specific practices within each process area were analyzed.

For each instance of processes two diagrams and SCAMPI report were prepared. Apart from the problems connected with the activities, which have been taken in the subprojects mentioned above and were still in progress by the self assessment time, there were noticed additional gaps to cover, e.g.:

- Requirements Management: the corporate standards and procedures are not always applied by project and product teams, instead other tools for requirements management are used,
- Project Planning: the corporate risk management standards and procedures are insufficiently implemented in projects and product development and maintenance centers,
- Project Monitoring and Control: the reporting standards and procedures are insufficiently implemented in projects,
- Supplier Agreement Management: agreements signed with suppliers not always require, that supplier must apply CL's rules and standards and sometimes not guarantee, that CL has possibility to fully monitor and control supplier's processes,
- Measurement and Analysis: the corporate policy of measurement and analysis requires in the scope of establishing, monitoring and measurement of business, product, quality, competence objectives to be coherent for all the organizational levels,
- Process and Product Quality Assurance: insufficient project effort is spent on pure QA activities like coaching, auditing, reporting,
- Configuration Management: the corporate standards and procedures are not always applied by project and product teams, instead other tools for configuration management are used.

For all the gaps and problems found corrective and improvement activities have been taken. It is expected, that IME is a useful tool for involving any person of an organization in process improvement activities, thus increasing their awareness, understanding and involvement and is a means to further educate people in an organization in the CMMI.

Weakness of the IME tool is the lack of possibility to consolidate data concerning different instances and that only incremental charts are accessible for CMMI levels.

4. Next Steps

Actually, the corrective and improvement processes are performed under OMD control. The consulting company has been chosen and it is expected that Orientation Meeting and Standards and Procedures Conformance Review will start soon.

It is expected that CMMI project will be finished to end of year 2005. Of course improvement process will never finish and will be a continuous process.

References

- [1] CMMI Product Team, Capability Maturity Model Integration (CMMI), v1.1, Staged Representation, CMU/SEI-2002-TR-004, Software Engineering Institute, Pittsburgh, PA, December 2001
- [2] Mutafelija, B., Stromberg, H., Systematic Process Improvement Using ISO 9001:2000 and CMMI, Artech House, Norwood, MA, April 2003
- [3] Mutafelija, B., Stromberg, H., ISO 9001:2000 – CMMI v1.1 Mappings, SZPG 2003 Conference Boston MA February 2003
- [4] De Smet M., Interim Maturity Evaluation based on Capability Maturity Model Integrated for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing, v1.1, http://www.man-info-systems.com/MIS_files/page0006.htm
- [5] International Organization for Standardization, Quality management systems – Fundamentals and vocabulary, ISO 9000:2000, ISO publication, December 2000
- [6] International Organization for Standardization, Quality management systems – Requirements, ISO 9001:2000, ISO publication, December 2000
- [7] International Organization for Standardization, Quality Management Systems – Guidelines for performance improvements, ISO 9004:2000, ISO publication, December 2000

Development of Safety-Critical Systems with RTCP-Nets Support

Marcin SZPYRKA

*AGH University of Science and Technology, Institute of Automatic
Al. Mickiewicza 30, 30-059 Krakow, Poland*

Abstract. Safety-critical systems require the utmost care in their specification and design to avoid errors in their implementation. Development of such systems may be supported by formal methods. The paper presents a subclass of timed coloured Petri nets called RTCP-nets that may be used for the modelling and analysis of safety-critical systems. Subclass of RTCP-nets is described shortly and some aspects of the modelling of safety-critical systems with RTCP-nets and implementation in Ada are also presented. (The work is carried out within KBN Research Project, Grant No. 4 T11C 035 24.)

Introduction

Critical systems are systems that may result in injury, loss of life or serious environmental damage upon their failure [5]. The high cost of safety-critical systems failure means that trusted methods and techniques must be used for development. For such systems, the costs of verification and validation are usually very high (more than 50% of the total system development cost). To reduce the amount of testing and to ensure more dependable products formal methods are used in the development process. A survey of some commonly used formalisms can be found in [3].

Petri nets are one of the most famous formalisms. They combine a mathematical representation with a graphical language. The presented approach uses RTCP-nets as modelling language for safety-critical systems. RTCP-nets (Real-Time Coloured Petri nets) are a subclass of timed coloured Petri nets (see [4]). The modifications defining this subclass were introduced in order to improve modelling and verification means in the context of analysis and design of embedded systems. RTCP-nets were defined to speed up and facilitate the drawing of Petri nets and also to equip Petri nets with capability of direct modelling of elements such as task priorities, over-timing, etc. RTCP-nets differ from timed CP-nets in a few ways. They use different time model, transitions' priorities and may be forced to fulfil some structural restrictions (see [8], [6]).

The presented formalism may be used for the design, specification, and verification of embedded systems. Especially, this technique has mostly been concerned with relatively small, critical kernel systems. An RTCP-net may represent an embedded system together with its environment. Therefore, it can be treated as a virtual prototype – one can check what actions the system performs in response to changes in the environment.

Another advantage of RTCP-nets is relatively simple transformation from a formal model into an implementation in Ada. Such an implementation is done with the use of

so-called Ravenscar profile. The profile is a subset of Ada language. It has been defined to allow implementation of safety-critical systems in Ada. Ravenscar defines a tasking run-time system with a deterministic behaviour and a low complexity.

1. RTCP-Nets

The definition of RTCP-nets is based on the definition of CP-nets presented in [4], but a few differences between timed CP-nets and RTCP-nets can be pointed out.

The sets of places and transitions of an RTCP-net are divided into subsets. Main places represent some distinguished parts (elements) of a modelled system, e.g. objects and main transitions represent actions of a modelled system. Auxiliary places and transitions are used on subpages, which describe system activities in detail.

The set of arcs is defined as a relation due to the fact that multiple arcs are not allowed. Main places may be connected to main transitions only. Each arc has two expressions attached: a weight expression and a time expression. For any arc, each evaluation of the arc weight expression must yield a single token belonging to the type that is attached to the corresponding place; and each evaluation of the arc time expression must yield a non-negative real value for an arc with place node belonging to the set of main places or 0 otherwise.

The time model used by RTCP-nets differs from the one used in time CP-nets. All types (colours) are considered as timed ones. Time stamps are attached to places instead of tokens. It is assumed that the time stamps attached to auxiliary places are always equal to 0. Any positive value of a time stamp describes how long a token in the corresponding place will be inaccessible for any transition. A token is accessible for a transition, if the corresponding time stamp is equal to or less than zero. For example, if the stamp is equal to -3 , it means the token is 3 time-units old. It is possible to specify how old a token should be so that a transition may consume it.

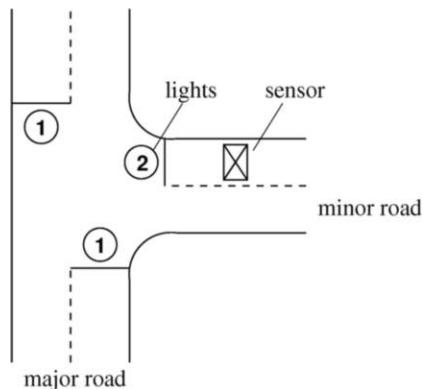


Figure 1. The simple road junction

Let's consider a traffic lights control system for the T-junction between a major and a minor road presented in Figure 1. The control system must ensure the safe and correct functioning of a set of traffic lights. The lights will be set to green on the major road and red on the minor one unless a vehicle is detected by a sensor in the readjust

ment (*Region* – this place is used to bound the number of vehicles), and places used to control how often new vehicles appear in input roadways (*Clock1*, *Clock2*). Transitions *Entry1*, *Entry2*, *Exit1* and *Exit2* model the traffic in the considered system.

Weight and time expressions of an arc are separated by @. Time expressions that are equal to 0 are not presented in the figure. Transitions *StartClock*, *Switch1*, *Switch2* and *Activate* have priorities attached that are equal to 1. Other transitions priorities are equal to 0.

Let the set of places be ordered as follows: $\{Driver, Lights1, Lights2, Sensor, MajorRoad, MinorRoad, Clock1, Clock2, Region\}$. The initial state may be presented as a pair of vectors (M_0, S_0) , where:

$$M_0 = (0, green, red, 0, 5, 5, 0, 0, 10), \quad S_0 = (0, 0, 0, 0, 0, 0, 0, 0, 0).$$

M_0 denotes the initial marking, while S_0 denotes the initial values of time stamps.

Dynamic aspects of RTCP-nets are connected with occurrences of transitions. An occurrence of a transition removes tokens from input places of the transition and adds tokens to output places. The values of tokens removed and added by an occurrence of a transition are determined by arc expressions. Only one token flows through each arc. New time stamps of output places are determined by time expressions of the transition output arcs. A transition is enabled if all input places contain proper values of tokens and also have proper time stamps, all output places are accessible, and no other transition with a higher priority value is trying to remove the same tokens. There are three enabled transition in the initial state: *Activate*, *Entry1* and *Exit1*. For example, the result of firing the transition *Exit1* is a new state (M_1, S_1) , where:

$$M_1 = (0, green, red, 0, 4, 5, 0, 0, 11), \quad S_1 = (0, 0, 0, 0, 2, 0, 0, 0, 0).$$

Analysis of RTCP-nets may be carried out with reachability graphs. Such a graph is finite for bounded nets, while the analogous graph for timed CP-nets is usually infinite. The reachability graph may be used to analyse the boundedness, the liveness and the timing properties of RTCP-nets [7].

2. Ravenscar Profile

The Ravenscar Profile (see [1], [2]) defines a simple subset of the tasking features of Ada in order to support safety-critical applications that need to be analysed for their timing properties. The requirement to analyse both the functional and temporal behaviours of such systems imposes a number of restrictions on the concurrency model that can be employed. The most important features of Ravenscar Profile are as follows:

- An application contains a fixed set of Ada tasks (processes), with the use of fixed priority scheduling. All tasks have to be declared at the library level and they never terminate – each task contains an infinite loop.
- The synchronisation or/and exchange of data between tasks is done by means of protected objects – the rendez-vous mechanism is not allowed. Only one entry for a protected object is allowed, and the guard of this entry has to be a simple Boolean variable. The maximum length of the entry queue is one.

- Implementation of periodic processes is possible using *delay until* operations.
- Protected procedures are used as interrupt handlers.
- The dynamic allocation, *requeue* statement, asynchronous transfer of control, *abort* statement, task entries, dynamic priorities, Calendar package, relative delays, *requeue* and *select* statement are not allowed.

The use of the Ravenscar profile allows timing analysis to be extended from just the prediction of the worst-case behaviour of an activity to an accurate estimate of the worst-case behaviour of the entire system. For more details see [1], [2].

An RTCP-net is as a starting point for the implementation of a system. Some more important parts of the Ada source code for the considered system are presented below.

Passive objects are usually implemented as protected objects without entries. A protected procedure is used when the internal state of the protected data must be altered, and a protected function is used for information retrieval from the protected data, when the data remains unchanged. *Lights1* and *Lights2* may be implemented as follows:

```
type Lights is (red, green);
protected type TLights (L : Lights) is
  procedure SetLights(L : in Lights);
private
  pragma Priority(Priority'Last);
  LState : Lights := L;
end;
...
Lights1 : TLights(green);
Lights2 : TLights(red);
```

The implementation of active objects depends on the role an object plays in the system. If such an object is used as an interrupt handler it is implemented as protected objects that handles the interrupt. An example of such an object is *Sensor*. The procedure *Activate* is identified as an interrupt handler by pragma *Attach_Handler*. The entry *Wait* is used by the *Driver* task.

```
protected type TSensor is
  entry Wait;
  procedure Activate;
  procedure Restart;
private
  pragma Attach_Handler(Activate, <interrupt_id>);
  pragma Interrupt_Priority(Interrupt_Priority'Last) ;
  SState : Boolean := false;
end;
...
```

Another active objects are usually implemented as one or more tasks. The number of tasks depends on the number of different cycles of activities such an object performs. If a token in the corresponding place contains an important information for the other active objects, an additional protected object is defined. Let's consider the object *Driver*. The token in the place *Driver* is used only to ensure the correct order of firing

of the transitions *StartClock*, *Switch1* and *Switch2* that are fired serially. In such a case, the active object is implemented as a single task:

```

task type TDriver (Pr : Priority; ST : Positive) is
  pragma Priority(Pr);
end;

task body TDriver is
  TimePoint : Ada.Real_Time.Time;
  SuspensionTime : constant Time_Span := Milliseconds(ST);
begin
  loop
    Sensor.Wait;
    TimePoint := Clock + SuspensionTime;
    delay until TimePoint;
    Lights1.SetLights(red);
    Lights2.SetLights(green) ;
    TimePoint := Clock + SuspensionTime;
    delay until TimePoint;
    Lights1.SetLights(green) ;
    Lights2.SetLights(red);
    Sensor.Restart;
  end loop;
end;

```

3. Summary

Some aspects of the development of safety-critical systems have been considered in the paper. It has been shown that development of such systems may be supported by RTCP-nets. The nets may be used both to model and to analyse the system properties. Moreover, some aspects of the implementation of such a system in Ada language have been also presented.

References

- [1] Burns, A., Dobbing, B., Romanski, G.: The Ravenscar Tasking Profile for High Integrity Real-time Programs. In: *Reliable Software Technologies, Proceedings of the Ada Europe Conference*, Uppsala, Sweden, pp. 263-275 (1998)
- [2] Burns, A., Dobbing, B., Vardanega, T.: Guide for the Use of the Ada Ravenscar Profile in High Integrity Systems. *University of York Technical Report YCS-2003-348* (2003)
- [3] Heitmeyer, C., Mandrioli, D.: *Formal Methods for Real-Time Computing*. *Jonh Wiley & Sons* (1996)
- [4] Jensen K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Vol. 1,2 and 3, New York, *Springer Verlag* (1996)
- [5] Sommerville, I.: *Software Engineering*. Pearson Education Limited (2004)
- [6] Szpyrka, M.: Fast and Flexible Modelling of Real-time Systems with RTCP-nets. *Computer Science* (2004)
- [7] Szpyrka M.: Reachability and Coverability Graphs for RTCP-nets, submitted to the International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)
- [8] Szpyrka, M., Szmuc, T., Matyasik, P., Szmuc, W.: A formal approach to modelling of realtime systems using RTCP-nets. *Foundation of Computing and Decision Science*, Vol. 30, No. 1, pp. 61-71 (2005)

ATG 2.0: the Platform for Automatic Generation of Training Simulations

Maciej DORSZ ^a, Jerzy NAWROCKI ^b and Anna DEMUTH ^c

^a *Projekty Bankowe Polsoft Sp. z o.o.,
60-965 Poznań, Poland*

e-mail: maciej.dorsz@pbpolsoft.com.pl

^b *Poznan University of Technology,
60-965 Poznań, Poland*

e-mail: jerzy.nawrocki@put.poznan.pl

^c *University School of Physical Education in Poznań
61-871 Poznań, Poland*

e-mail: demuth@awf.poznan.pl

Abstract. Automation based on code generation from diagrams, usage of advanced components, or automated testing based on the GUI is becoming a standard in the software production process. The next stage is an automatic generation of a training simulation. Thanks to it, the end-users participating in a training can work with their system version, and after finishing the course they may take such a simulator home or to their office and keep on practicing. Automatic Training Generations (ATG) system is designated for the automatic generation of the final system simulations. ATG allows to generate a training for the given client, regarding his/her particular system version and individual system settings. Moreover, ATG 2.0 makes possible to use differences mode, which presents only changed system features.

Introduction

Models and standards such as ISO 9001:2000, Prince II or XPrince are used not only to improve software production processes, but they also put stress on the role and importance of the client [2], [4], [9]. The final product is designed for the client and he/she will use it. One of the final stages in the software production process are staff training. This step means, for most of that staff, the first contact with a new product, and unfortunately it quite often brings to mind or elicit negative feelings. For many people attending the training a new tool is associated with stress connected with a new methodology; a fear that colleagues will absorb new material faster and the conviction, based on experience, that the time gap between the training and the final product deployment will be long enough to have the knowledge acquired during the course forgotten.

The solution to the aforementioned problems may be a training based on a simulation of the final application. Then every training participant works on their own computer and is able to adjust learning speed to his/her own abilities. After a training the willing can continue the training even at their own home.

DiMon application developed in Projekty Bankowe Polsoft Sp. z o.o. has been deployed in more than 10 financial institutions which means a lot of training for many end-users groups. But, it is worth mentioning that the clients have dedicated parameters settings, which change functionality as well as system GUI. On the basis of these needs an idea of a system, which could be used to generate final application simulation was introduced. Therefore, ATG (automated training generation) system was created. ATG allows one to generate training simulations in an automated way.

In Section 1 a few positive aspects of using ATG systems are mentioned. Section 2 briefly presents system architecture with particular stress put on three main system modules. These modules are described in Sections 3, 4, and 5. Section 6 presents *differences* mode.

1. Common Training Problems

The need to use a simulator instead of a training in a life environment was noticed a long time ago, among other things in air forces or motorization [6]. Nowadays many simulators which help one to acquire a given domain knowledge are created. They also reduce training cost, make it easier to have access to the tools, and increase students security. ATG allows to overcome following problems [3]:

Technical Problems

Technical problems are unfortunately an indispensable element of many training. Usually the company which delivers the product leads a training at the client site, what is naturally very convenient for the client. However, a trainer uses then application instance working at client environment (his/hers servers, databases, network architecture, etc.). In the case of problems a person leading a training is dependent on the client staff.

Reducing the Time Gap between a Training and Final System Deployment

Training are usually lead at a test system version. It means for the end-users that the next contact with a given system will be after its delivery. Till this time they won't be able to use this application, so they cannot repeat material covered during the training. Preparing a training as a collection of HTML pages, which can be just copied to a hard disk without an installation necessity, each student can use such a simulator on their computer and use it even the following day after the training.

Training will be Prepared Carefully

XML templates with examples (simulational use-cases) allows to generate task contents for the training. Is quite common that a trainer generates examples ad-hoc during the course. However, some of those examples may be not particularly interesting or didactic. Moreover, training participants often try to make notes and this may be stressful for them and also lead to slowing down the course speed.

Equal Data

Because a simulator does not use database, therefore, course participants can input the same data. Using a test version of a system some data must be unique, for example, considering a bank application the transaction number must be unique. Therefore, the leader must coordinate the "generation" of such numbers and ask students to work only on transactions they added.

Individual Speed

Every course participant do exercises in his/her own speed. Therefore, listing to trainer's instructions in a traditional approach the course rate is usually adjusted to the slowest working person. In the proposed solution, people who learn faster can do the exercises quicker without losing their concentration. This approach reflects one of the newest trends in the teaching methodology called: the learner-centered approach [1], [5].

2. System Architecture

Figure 1 presents the three main system modules. Arrows connecting the modules show the sequence of data processing. The output of one module is the input of the other, therefore these modules can be called processes [4].

First module on the basis of automatic testing tool, which executes test cases, saves subsequent application pages as HTML files. Then to these pages static elements must be added such as: pictures, .css files, .js files etc.

Second module uses use-case diagram in order to prepare a simulation context.

Last, but not least, module transforms saved HTML pages according to simulational use-cases. This module changes saved pages to make user convinced that he/she uses a real application and not just a system simulation. Therefore, it adopts links, but also blocks not used fields and buttons etc. Additionally, it adds the possibility of using *help* during solving tasks.



Figure 1. Main system modules

3. Saving HTML Pages

First step to prepare a training is based on preparing use-cases for automatic testing. They will also be used to save subsequent HTML pages. For DiMon QALab technology, based on Compuware tools, was used. But one can use any tool which will allow to save subsequent HTML pages.

4. Preparing Roles and Use-Cases

Stage number two starts with preparing a use-case diagram in ArgoUml and saving it as .xmi file [3]. On the basis of it ATG will generate the framework of HTML pages for a training simulator regarding division into roles and use-cases. Thanks to such a hierarchy end-user may focus their attention only on the use-cases they will need in their work. These pages may be also prepared manually or test use-cases may be used to generate roles and functions automatically. In Figure 2 framework pages generated on the basis of the sample use-case diagram are presented.

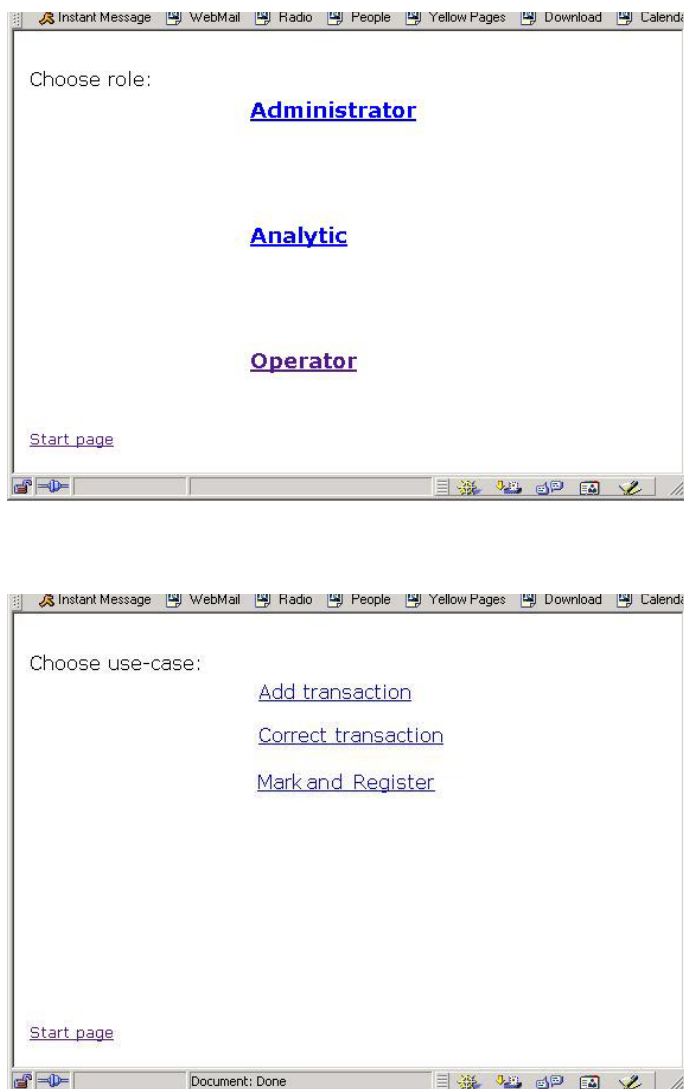


Figure 2. Pages generated on the basis of the use-case diagram

5. Simulation Generator

The module uses simulational use-cases written in XML and saved HTML pages to prepare the final simulation [3]. It adds JavaScript used for events on fields required for finishing a given task. If a user places an incorrect value, the system simulation will notify the user. Moreover, transformation process blocks fields, buttons, and links which values needn't be changed. Then the user who tries to fill such a field will be presented a message informing that value of this field shouldn't be modified [3]. Finally, only positive filling of required fields allows to go to the next step or to successfully finish a given use-case.

Such prepared pages make the real simulation, which allows end-user to choose a role and than use-cases [3]. Next, the system presents general task description. For example: ‘*Your task is to mark one transaction as suspected and then register all the transactions*’. Then, the end-user uses the set of HTML pages as if he/she were connected to the real system. Moreover, ATG cares about data validity and sequence of chosen links/buttons. The user can choose the help button to for instructions.

6. Differences Mode

The Differences mode allows end-users to focus their attention only on the differences between the newest version and one of the previous system versions. Presented differences cover added simulational use-cases and differences in already existing ones, such as: use-cases descriptions (marking added, removed, deleted words), subsequent steps, and task details.

Figure 3 presents summary of differences between the newest and the chosen by the end-user system versions. Links allow to practice a given-use case or to be shown differences between descriptions, subsequent steps or task details. Figure 4 shows task description differences; words marked in green means added words. *Task details* link shows the differences in filling task particular data.

The newest system version in comparison to 1.0build98.

Role: Operator

Simulational use-case	Differences		
	Task description	Subsequent steps	Task details
Mark and Register	show	show	show

Role: Administrator

Simulational use-case	Differences		
	Task description	Subsequent steps	Task details
Export transactions	added	added	added

[Start page](#)

Figure 3. Differences summary

7. Summary

This article presents the main ideas concerning ATG system. ATG platform allows to generate a training based on the simulation of the application in an automatic way. Many benefits may be taken from using ATG both on the client and software company sites. Thanks to the described solution a training has a chance to be prepared carefully, avoiding any technical problems. Because of an individualised way of prosing new knowledge students can learn at their own rate not worrying that others may acquire

material faster or slower. Using ATG allows students to take the simulation home or office and continue learning. Moreover, *differences* module makes it possible for users who already know the system to focus only on the changes between subsequent versions.

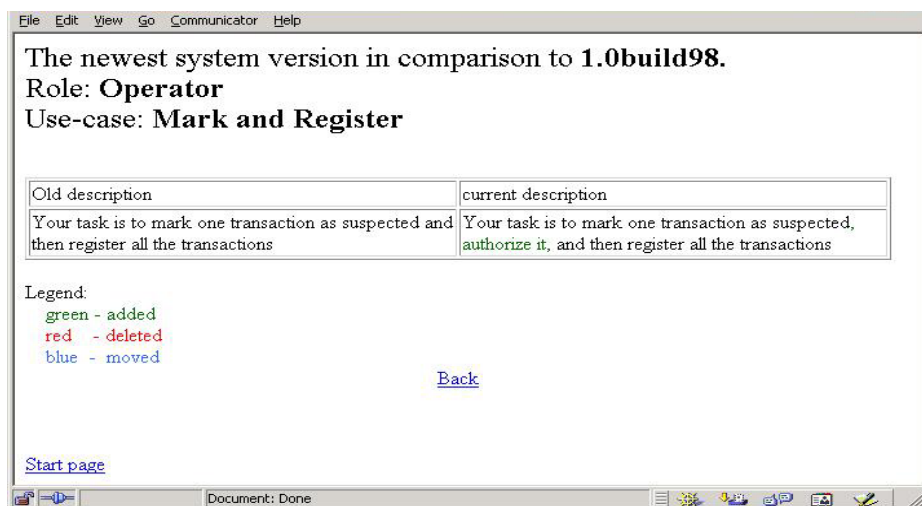


Figure 4. Task description differences

References

- [1] Carter, R.: *Introducing Applied Linguistics*. The Penguin Group: London (1993)
- [2] CCTA, *Managing Successful Projects with PRINCE 2*, The Stationary Office, London (2002)
- [3] Dorsz, M. Nawrocki J., Demuth, A.: *ATG: Platforma do automatycznego generowania symulacyjnych systemów szkoleniowych*, Zeszyty Naukowe Wydziału ETI Politechniki Gdańskiej, Technologie Informacyjne 2005
- [4] ISO 9000:2000, ISO, Geneva (15.12.2000).
- [5] McCombs, B. L., Whisler, J. S.: *The Learner-Centered Classroom and School: Strategies for Increasing Student Motivation and Achievement*. Jossey-Bass: San Francisco (1997)
- [6] http://encarta.msn.com/encyclopedia_761578212/Flight_Simulator.html
- [7] <http://www.software.com.pl/konferencje/sqam/index.php?page=w12>
- [8] <http://en.pbpolsoft.com.pl/aktualnosci/newsy/2004/listopad/konfsjsi.xhtml>
- [9] <http://www.cs.put.poznan.pl/jnawrocki/io/0304/04ioc-inz.ppt>

RAD Tool for Object Code Generation: A Case Study

Marcin GRABOŃ^a, Jarosław MAŁEK^a, Marcin SURKONT^a,
Paweł WOROSZCZUK^a, Rafał FITRZYK^a, Andrzej HUZAR^a, Andrzej KALIŚ^b

^a*Huzar Software,*

ul. Tczewska 17, 51-429 Wrocław, Poland

e-mails: {grabon, malek, surkont, woroszczuk, fitrzyk, andrzej.huzar}@huzar.pl

^b*Wrocław University of Technology, Institute of Applied Informatics,*

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

e-mail: andrzej.kalis@pwr.wroc.pl

Abstract. This paper presents the *Class Editor* tool. It is a software development environment that provide automated support for generating Delphi language code from class hierarchy defined either by developers directly or by reading the source data from XML files. We believe, that those classes affect the flexibility of generated code, program comprehension, support round-trip engineering, and program maintenance. In addition this tool can convert classes to highlighted HTML property name documentation, and generate XML Schema files, as well as ready to use templates of read/write procedures from/to databases, providing mechanisms for accessing and storing data. Moreover, it can generate consistent class types hierarchy from example XML data files. Given examples have shown that the tool is useful, and important to us. According to the best knowledge of the authors, such tool has been introduced in this paper for the first time.

Introduction

Keen competition throughout the computer industry, and time to market often determines success, therefore all software engineers use tools and environments (e.g. Borland Delphi [5]). Tool support can have a dramatic effect on how quickly software product can be done and on the quality of the result [3]. Code generation techniques are concerned with translation of high-level specifications into code. These techniques can improve productivity by allowing developers to specify software behaviour at a high level of abstraction, leaving the code generator to manage the implementation details [6]. Every software system that is being used needs to be maintained. Software is only finished when it is no longer in use [9]. The number of releases for a new application that needs to stay ahead of the competition by adding features (e.g. law changes, tariff codes, tax due and so on) every week may be very high. There are three main reasons for changing software: (1) bug fixing, (2) stay competitive, and (3) legal or technical reasons. Every manager knows that a non-perfect but acceptable program version on time is better than fine-tuned one year later (so called „Good Enough Software” [10]). There is a quality incentive, but it only leads to the acceptability point at which remaining deficiencies do not endanger its usefulness to the market [4].

Therefore, taking the above into account we present a self-designed tool, called *Class Editor*, that helps us in code generation and documentation preparation for

defined classes structure, and in consequence gives us chance to built software on time, easily maintained and comprehensible.

The Huzar Software company for many years develops software supporting the process of preparing required documents for customs clearance. Very frequent changes in customs regulations, especially everyday data changes in the customs tariff (sic!) and rather frequent changes in the structure, cause that the software development should be aided by very efficient tools. We have designed and implemented our own tool supporting programming which collaborates with Delphi creating “super RAD”.

In our fifteen year experience in writing software, the main thing we have learned about requirements is that: “Requirements always change”. Therefore, we must write our code to accommodate change, and we should stop beating ourselves up (or our customers, for that matter) for things that will naturally occur [7], and we should use special development tools that deal with changing requirements more effectively.

1. Fundamental Ideas of our Approach to Programming

Software created in our company is guided on processing documents, sometimes very complex (e.g. Single Administrative Document – SAD; INTRASTAT). It shows several years of our experiences, that the programmer should use of tools fulfilling following conditions:

- the entire document should be present in the memory and should be represented as an object (not record structures, because they don't have methods with which they can store themselves to memory or to the different kind of disk files),
- the memory management should be invisible for the programmer (except of creation and releasing), irrespective on the complexity of the class hierarchy,
- the objects should have the defined useful methods, e.g. sorting, comparing, finding, copying, adding, deleting as well as clearing of objects,
- the objects should have mechanisms for accessing and storing data,
- access to RTTI information is rather slow, therefore all objects should have specific methods that get size of object, property or field name and indexes (as class methods operate on class types themselves),
- the possibility of checking if the field has a value (it is important in so called NULL problem [8]), and if the field is stored.

2. Description of Class Hierarchy Used in *Class Editor*

The unit called `HSBaseClasses.pas` (so called “engine”) contains declarations of all the classes mentioned below. The *Class Editor* use these class types during generation of source code. We have defined more than 30 classes, therefore corresponding picture of class diagram is very big and in consequence is omitted. Number of properties and methods is given to show the complexity of our base class hierarchy.

The mother of our class hierarchy is `THSField` that descends directly from `TPersistent`. Because all further introduced objects are descendants of `THSField`, they are all streamable. The class has only two properties: `AsVariant`, `AsString`,

and a few virtual methods (as we see the names are self-explanatory): `Create`, `ClearData`, `CopyData`, `EditData`, `LoadFromStream`, `SaveToStream`, `Compare`, `SaveAsText`, `LoadAsText`, `SaveAsXML`, `LoadAsXML`. As mentioned above, all simple data types have to be represented in the form of objects. Therefore simple types such as `String`, `WideString`, `Boolean`, `Byte`, `SmallInt`, `Integer`, `Cardinal`, `Int64`, `Double`, and `DateTime` are represented by the following objects: `THSFieldString`, `THSFieldWideString`, `THSFieldBoolean`, `THSFieldByte`, `THSFieldSmallInt`, `THSFieldInteger`, `THSFieldCardinal`, `THSFieldInt64`, `THSFieldDouble`, `THSFieldDateTime`, respectively. All of these “simple” objects descend from `THSField`. They have only one property named `Value` with a corresponding type, and methods that override corresponding virtual methods from `THSField`.

The next base class `THSBaseObject` = `class(THSField)` is more complex, and has 36 public methods and two properties defined (`Parent` – handle to parent class; `HasValue` – is `TRUE` when the value of field of given index is not `NULL`). The methods worth to mention (except those overridden from the superclass) are: `SaveToFile` (`LoadFromFile`) – it saves entire object to (loads from) given kinds (binary, txt, xml, RES, compressed, encrypted) of files; `TextDescription` – as a result return string that contains name of fields of the object. Useful in automated generation of edit forms, where labels of edited fields have the same caption as declared in the object. Of course, in *Class Editor*, the user may introduce meaningful names for form's labels, and in this case the `TextDescription` returns captions of all labels (using the `EditNameForFieldIndex` method).

The class `THSObject` = `class(THSBaseObject)` is very important, because among others this one can be accessed in *Class Editor*. It means that all user defined classes are descendants of `THSObject`. The class have 29 methods and 3 properties. For example, method `HasDefaultValue` touches the `NULL` problem. If we compare two objects this method replaces `NULL` values by corresponding default values (e.g. `integer(NULL)=0`); `HasFeats` – returns `TRUE` if in the given object field has the specified value; `CopyFieldData` – values of given fields are copied to other object fields; `ClearFieldsData` – clears given fields; `ClearFieldsDataExcept` – clears all fields except given ones. Some methods are declared as class methods (`TypeOfFieldIndex`, `SizeOfFieldIndex`, `NameOfFieldIndex`, `IndexOfFieldName`, `EditNameForFieldIndex`, `StoredField`), and operate on a class reference instead of an object reference (so class methods are accessible even when you haven't created an object). We see from self-explanatory names of methods why these six methods are declared as class methods.

The user of *Class Editor* is able to create class hierarchy not only using the above `THSObject` but also `THSListOfObjects`, `THSListOfWeakRefToObject`, `THSListOfRefToObject`, `THSCollectonOfObjects` that are descendants of `THSBaseList` (with 23 methods that operate on lists), and others not described here. Methods gathered in class `THSBaseList` give us several standard list procedures and functions such as counting, adding, deleting, clearing, moving, appending and sorting. The special `THSListOfWeakRefToObject` class gives us possibility to create list of weak references to other objects. Weakness of the reference means that freeing an object does not cause reference loss to that object (we only mention that we have also so called strong reference – `THSListOfRefToObject` in that case releasing an object forces reference clearing).

3. Class Editor Interface

The main window of *Class Editor* consists of four parts: the main menu, the navigator for class hierarchy/documentation, and source code/documentation panel [2]. *Class Editor* is able to save all his work to *.hsp project file that contains the documentation together with history of changes, last used projects, paths to source files etc.). The most important function shortly described in the paper are given as function of *Types* menu [2].

Class Editor for any type descending from *THSObject* provides XML Schema file generation (xsd file), which is compatible with <http://www.w3.org/2001/XMLSchema> format. Schema file contains all fields of objects, including weak references, which are represented as a textual path. All field-length restrictions are preserved, as well as field format data. Schema file allows users to prepare and validate files created by external applications, that later can be read by Huzar Software applications. Schema files can also be prepared specific for Microsoft Excel 2003, which has some limitations on importing this kind of file.

4. Practical Code Generation Example

Class Editor makes possible automatic creation of the “documentation of types” by analysing the content of XML files. By documentation of types we understand a tree structure of types along with the relationships between them. Editor is able to recognise all simple types such as string, integer, double, etc. and user-defined types (the objects as well as the list of objects). The documentation of types is the result of content analysis, and from this one editor is able to generate the class hierarchy and conforming source code that includes implementation of classes shortly described in the previous section. Moreover, if class hierarchy is modified (new class types added, sizes of fields changed, etc.), the editor is able to make update of the documentation. All differences are recorded by the editor (in form of date, time, and automatically generated short description of changes appearing).

Naturally, the actual documentation may also be updated, if source data contained in xml files have been altered. *Class editor* will analyse again the xml files, and will indicate to user differences between content of files and documentation (inside the documentation navigator). When user recognizes that these differences may be accepted then we can update the class hierarchy as well as source code. Proceeding in this way we are certain that the source code is consistent with documentation. In addition, we can introduce to documentation own information, such as required rules, remarks or description of fields meaning, and we can export this documentation to HTML format. It is easily noticed, that *Class Editor* is very strong tool permitting to create documentation on basis of class types and in the reverse direction can create class types definition (“round-trip engineering”).

The above presented mechanism have been used to creation of types of customs tariff, based on the original source data in the form of pure xml files (c.a. 1.5GB in size). *Class Editor* have analysed these files and then have created documentation of types (more than 200 types). Whole process lasted less than 20 minutes (on IBM PC with clock 900 MHz, and memory – 256 MB), and generated source code has more than 12000 lines of Delphi language code. We are careful that manual creating code for that number of types is almost impossible.

Unfortunately data contained in pure XML files (without XSD) not always exactly reflect types in comparison to paper documentation. For example, the size of some string type can be determined if the field of that type exists in the source data. In several cases some fields do not contain any data or simply are skipped, so user manual inspection is necessary (about two hours of time were engaged).

When we have prepared source code together with the features of our “engine” class types described above then we are able to map them into database structures. It is possible because of class methods existence (we can traverse objects and read all information interesting for us, and then implement access to database procedures). In our examples of customs tariff we have built the database in master-detail-detail structure (because procedures for accessing and storing data operate on `THSObject`, `THSListOfObjects` only). We remind, that modifications of data sources (or manually introduced modifications to class hierarchy using editor) are able to change our source code, and in consequence change database structure in run-time (by proper table restructure mechanisms) without changing source code responsible for accessing database. It is also evident that we are able to encrypt and compress data in straightforward manner.

Given example have shown that the tool is useful, and important to us because of significant manner it accelerates the process of the creation of software. Moreover, ready to use source code can be used without any changes by other members of the company that have to access to customs tariff.

5. The Importance of Documentation and Code Comprehension

As stated in [1] “Programming environments that support evolutionary software development must include tools that help programmers understand complex programs”, therefore we believe that these “old” ideas should be extended further so that the documentation associated with class structure design is manipulated in the similar way as software documentation. Thus our class editor should reinforce the fact that programs and their documentation are hierarchical compositions and should provide facilities to edit, execute, debug and understand them. So the class editor generates documentation based on class definition as an on-line documentation browser (in highlighted HTML), so navigating the dependencies in class definition is as easy as surfing the web. But the most important is that the documentation is directly extracted from the sources, which makes it much easier to keep the documentation consistent with the source code.

One of the main reasons the code reuse is so hard is the fundamental law of programming: “It’s harder to read code than to write it”. The editor of classes permits to keep sure programming characteristic style which facilitates understanding programme superbly.

6. Conclusion

This paper presents the *Class Editor* tool. It is a software development environment that provides automated support for generating Delphi language code from class hierarchy defined either by developers directly or by reading the source data from

XML files. The presented tool is useful, and important to us because of significant manner it accelerates the process of the creation of the software.

References

- [1] Ambras J.P., Berlin L.M., Chiarelli M.L., Foster A.L., O'Day V., Splitter R. N.: Microscope: An integrated program analysis toolset, *Hewlett-Packard Journal*, 39(8), pp. 71–83, August 1988
- [2] Class Editor screenshots, <http://www.huzar.pl/ClassEditor/ClassEditor.html>
- [3] Harrison W., Ossher H., Tarr P.: Software Engineering Tools and Environments: A Roadmap, in *The Future of Software Engineering*, A. Finkelstein (Ed.), *ACM Press* 2000
- [4] Meyer B.: The Grand Challenge of Trusted Components, in *25th International Conference on Software Engineering*, Portland, Oregon, May 2003, pp. 660-667
- [5] Pacheco X., Teixeira S.: Delphi 6 Developer's Guide, *Sams Pub*, 2002
- [6] Schumann J., Fischer B., Whalen M., Whittle J.: Certification Support for Automatically Generated Programs, *HICSS'03*, January 2003
- [7] Shalloway A., Trott J.R.: Projektowanie zorientowane obiektowo – wzorce projektowe, *Helion*, Gliwice, 2001
- [8] Subieta K., Kambayashi Y., Leszczyłowski J., Ulidowski I.: Null Values in Object Bases: Pulling Out the Head from the Sand, 1996, <http://sherry.ifi.unizh.ch/subieta96null.html>
- [9] Swanson B.: The dimensions of maintenance, In *Proc. of the Second International Conference on Software Engineering*, pp. 492–497, 1976
- [10] Yourdon E.: When Good-Enough Software is best, *IEEE Software*, Vol. 12, no. 3, May 1995, pp.79-81

KOTEK: Clustering Of The Enterprise Code¹

Andrzej GAŚIENICA-SAMEK^a, Tomasz STACHOWICZ^a,
Jacek CHRZĄSZCZ^b and Aleksy SCHUBERT^b

^a *ComArch SA,*
ul. Leśna 2, 02-844 Warsaw, Poland
^b *Institute of Informatics, Warsaw University,*
ul. Banacha 2, 02-097 Warsaw, Poland

Abstract. Development of large code bases is extremely difficult. The main cause of this situation is that the internal dependencies in a large code body become unwieldy in management. This calls for methods and tools that support software development managers in maintaining properly ordered connections within the source code. We propose a method and a static module system KOTEK to facilitate high and medium level management of such source code dependencies. The system enforces all dependencies to be clearly declared. Since KOTEK is also a build system, it automatically enforces these declarations to be up-to-date. Moreover, KOTEK allows advanced software engineering constructions like parametrisation of large code fragments with respect to some functionality.

Introduction

Big source code bases are extremely difficult to develop and maintain. Thus, a proper management of the code is needed [12]. There are various ways to organise the code. In object-oriented languages, the most basic ones are objects (or classes). The objects or classes are usually considered as low-level units, though, so they are grouped in components, packages or modules.

The power of the organisation mechanism depends on the way the grouping affects the code and is imposed on the code. For instance the tools which are based on UML or Semantic Web ontologies provide grouping in the design stage of software production but are weakly enforced in the development and maintenance stages. Moreover, they do not encourage comprehensive arrangement of construction blocks and so complicated diagrams are commonly encountered.

Moreover, the flexibility of these design standards and programming language grouping constructs like packages make it easy to build circular dependencies. The experience in software development shows that circular dependencies cause problems [13], [6] so the DAG-based coding pattern occurs often in project design guidelines [10], [9], [4]. Cyclic dependencies are regarded as a strong factor in measures of code complexity [14] especially when maintainability of the code is of the main interest [8]. Moreover, the presentation of code dependencies in form of a DAG has already been used in the context of support for maintainability [2].

¹ This work was partly supported by KBN grant 3 T11C 002 27.

We propose a system KOTEK which assists in maintaining the code structure and in organising knowledge within a software project. It ensures the match between the description of the organisation and the code since it is a tool that builds the final application. All kinds of dependencies are based on the tree or DAG structure here. Moreover, we impose the rule that each component may consist of at most seven items [11], [5].

The basic unit of code organisation in KOTEK is called a module. KOTEK has two perspectives of code organisation. They correspond to a different basic source code organisation activities. The first one, *vertical*, allows to abstract knowledge about the modelled fragment of the real world. The second one, *horizontal*, describes functional dependencies between the abstracted notions. This division separates two basic modes of thinking. The first one focuses on the internal structure of the defined computing notion, and the second one focuses on the relations with other pieces of the software. Other Java module systems did not consider explicitly this code organisation facets [3], [7], [1].

As KOTEK is a build tool, it is related to ant, make and maven. The main difference is that these tools operate on files only while KOTEK performs semantical checks. Thus, it gives an additional control power for a code manager.

1. Gentle Introduction to KOTEK

In this section we describe the most important aspects of using the KOTEK tool through consecutive refactoring of an example of a simple database client application.

KOTEK builds the application making sure that all dependencies are declared and that their structure follows the structure described in Section 1. The invocation of KOTEK in the root directory of the project, makes it recursively build all components and combine them (link) into the resulting object file. The order of the building process and dependencies between modules must be described in the file `.kotek`, which is located in the project's root directory.

The first example. In this section we use the simplest version of our sample application. Its main `.kotek` file can be seen in Figure 1 left.

The first line of this file says that in order to build our project, one needs external libraries `JDBC` and `Swing`. Next, one has to build the module `DataModel` using `JDBC`, the module `UI` using `Swing`, and `Logic` using `Swing` together with just built `DataModel` and `UI`. The final *product* of our code is the module `Logic`, which provides a class with the main method.

Apart from being a building instruction, the `.kotek` file provides an overview of the main dependencies of the project which helps in understanding of the code.

Since `DataModel`, `UI` and `Logic` may be large pieces of code, they can also be divided into submodules and KOTEK can be used to manage the order of their building and their dependencies. We assume that larger modules lie in the corresponding subdirectories and each subdirectory contains the local `.kotek` file. For example, in the `Logic` directory, this file may look like in Figure 1 right. The `Logic` component contains 3 sub-modules: `DataManip`, `UILogic` and `App`.

Note that our modules `DataModel` and `UI` are treated inside `Logic` as external ones and the implementation of `Logic` has no access to their internal details.

uses JDBC Swing	uses DataModel Swing UI
build DataModel: JDBC	build DataManip: DataModel
build UI: Swing	build UILogic:
build Logic: DataModel Swing UI	DataModel DataManip Swing UI
	build App: Swing UILogic
return Logic	
	return App

Figure 1. Files Root/.kotec and Root/Logic/.kotec of the sample application

The hierarchical structure of .kotec files permits a person who wants to learn the code (e.g. a new developer) to read it in a needed level of details and only in the branches that are interesting at the moment.

Abstraction and Programming with Variants

Sometimes, almost identical code is used in several places of the whole project. This code must be placed in a separate organisational unit. This is done by abstraction. As the way the code is used in different places may differ, it is useful to have more than one run-time component derived from a single piece of the source code (for example, it is the case when one wants to provide several versions of the application, for different graphical environments). In KOTek, such multiple *products* of a single piece of the source code are called *views*. Each view may have different dependencies, as it is the case in the final version of our example in Figure 2.

File: Root/.kotec

```

uses JDBC Swing Forms

build DataModel: JDBC

absbuild UI: Swing Forms

let UICommon=UI create Common ()
let UIJ2SE=UI create J2SE (Swing)
let UIDotnet=UI create Dotnet (Forms)

absbuild Logic: DataModel UICommon Swing UIJ2SE Forms UIDotnet

let AppJ2SE=Logic create AppJ2SE(DataModel UICommon Swing UIJ2SE)
let AppDotnet=Logic create AppDotnet(DataModel UICommon Forms UIDotnet)

return AppJ2SE AppDotnet

```

File: Root/UI/.kotec

```

uses Swing Forms

build Common:
build Utils:
build J2SE: Utils Swing
build Dotnet: Utils Forms

return Common J2SE Dotnet

```

```

File: Root/Logic/.kotek

uses DataModel UICommon Swing UIJ2SE Forms UIDotnet

build DataManip: DataModel
build UILogic: DataModel DataManip param(UICommon contr Common) as UI
let UILogicJ2SE=UILogic(UIJ2SE as UI)
let UILogicDotnet=UILogic(UIDotnet as UI)
build AppJ2SE: Swing UILogicJ2SE
build AppDotnet: Forms UILogicDotnet

return AppJ2SE AppDotnet

```

Figure 2. Two versions of UI

In the example, we replace a single UI module from Figure 1 by two modules: UICommon and UIJ2SE. The first one provides only the abstract window interface used in our application. The interface can be understood as a Java package containing only class interfaces. The second module, UIJ2SE, provides the implementation of the abstract interface, based on Swing. Both modules are then passed on to the Logic component.

Moreover, we add a .NET frontend based on Forms to our application. Multiple views are used in two components in this version of our application. First, the two related modules, UICommon and UIJ2SE, have been joined into a bigger component UI, which got the third sub-module Dotnet, implementing the Common contract using Forms. The code that is the same in J2SE and Dotnet has been extracted to the module Utils. Apart from the latter, the other three modules are exported as three products of the UI components.

The Logic component changed accordingly: the UILogic is based on the common interface as before, and there are now two final modules AppJ2SE and AppDotnet, depending on suitable graphic toolkits and instantiated UILogic.

In the main .kotek file the dependencies of the modules UI and Logic are listed twice. The first time, in the `absbuild` command (a shorthand for *abstract build*), which causes a recursive build of the component but without the final linking phase. The second time, they are used as arguments of the `create` command which performs the linking. Note that by analysing the dependencies in the .kotek file alone it can be seen that the J2SE version of the application does not depend on Forms and that the Dotnet version does not depend on Swing.

2. Technical Overview

The KOTek tool is not limited to a particular programming language, even though we specifically thought of needs of large Java projects while designing it. It consists of four language layers, two of which are intermediate and hence practically invisible for users. These are (N) native (object files) e.g. Java, (M) low-level abstract (.ms files) invisible, (L) linker instructions (.cc and .ld files) invisible, (K) .kotek files.

The (M) level provides the detailed description of the interfaces of modules that define dependencies outlined in Section 1. The description language is independent

from the source language. The (L) level is the list of instructions for the linker, connecting formal parameters of modules to their actual dependencies. Formally, it just binds to new names the applications of functions to arguments.

The input for KOTEK is the list of object files of the module's dependencies, together with their contracts (`.ms` files) and type sharing information between the dependencies (in the `.cc` files). The output is the final object code and a pertinent `.ms` file with interface specification. The information is processed between the four layers as follows:

- **native reader** compiles source code only modules (without `.kotek` files) and derives `.ms` files for them,
- **native linker** links the object code of the submodules into the object code of the module, according to the linker instructions in the `.ld` file,
- **linker** links the `.ms` files of the submodules into the `.ms` file of the module, according to the linker instructions in the `.ld` file,
- **KOTEK main** transforms the input. `.cc` file and `.kotek` into `.cc` of each sub-module, runs KOTEK recursively, then builds the `.ld` file and calls the linkers to build the resulting `.ms` and object files.

Note that only the two first transformations are language specific and hence in order to use the KOTEK method for other programming languages, one only has to provide these two. Note also that it is possible to implement only part of the functionality for a given programming language (for example without abstract modules and views) and still benefit from other advantages of KOTEK.

3. Conclusions

Prototype. The KOTEK method is actively used in ComArch research laboratory to manage an actively developed Ocean GenRap business intelligence platform prototype, the project of about 210 000 lines of mostly Java code. It is managed using about a 100 `.kotek` files of total length of 1700 lines, so less than 1%. Even though not all features presented in the paper are implemented in the KOTEK prototype, it already proves to be a great help in learning the code by new developers and in managing the code by component owners.

References

- [1] Davide Ancona and Elena Zucca. True Modules for Java-like Languages. In *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 354-380, London, UK, 2001. Springer-Verlag.
- [2] Liz Burd and Stephen Rank. Using Automated Source Code Analysis for Software Evolution. In *1st IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2001)*, 10 November 2001, Florence, Italy, pages 206-212, 2001.
- [3] John Corwin, David F. Bacon, David Grove, and Chet Murthy. MJ: A Rational Module System for Java and its Applications. In *Object-Oriented Programming, Systems, Languages & Applications*, 2003.
- [4] Compuware. Optimal advisor supersedes the Package Structure Analysis Tool. Technical report, JavaCentral, 2005.

- [5] Jean-Luc Doumont. Magical Numbers: The Seven-Plus-or-Minus-Two Myth. *IEEE Transactions on Professional Communication*, 45(2), June 2002.
- [6] Martin Fowler. Reducing Coupling. *IEEE Software*, July/August 2001.
- [7] Yuuji Ichisugi and Akira Tanaka. Difference-Based Modules: A Class-Independent Module Mechanism. In *ECOOP '02: Proceedings of the 16th European Conference on Object-Oriented Programming*, pages 62-88, London, UK, 2002. *Springer-Verlag*.
- [8] Stefan Jungmayr. Testability Measurement and Software Dependencies. In *Software Measurement and Estimation, Proceedings of the 12th International Workshop on Software Measurement (IWSM2002)*. Shaker Verlag, 2002. ISBN 3-8322-0765-1.
- [9] Kirk Knoernschild. Acyclic Dependencies Principle. Technical report, Object Mentor, Inc., 2001.
- [10] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
- [11] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81-97, 1956.
- [12] A. Podgurski and L.A. Clarke. A Formal Model of Program Dependencies and Its Implications for Software Testing, Debugging, and Maintenance. *IEEE Transactions on Software Engineering*, 16(9):965-979, 1990.
- [13] Barry Searle and Ellen McKay. Circular Project Dependencies in WebSphere Studio, *developer Works*, IBM, 2003.
- [14] Lassi A. Tuura and Lucas Taylor. Ignominy: a tool for software dependency and metric analysis with examples from large HEP packages. In *Proceedings of Computing in High Energy and Nuclear Physics*, 2001.

Inference Mechanisms for Knowledge Management System in E-health Environment¹

Krzysztof GOCZYŁA, Teresa GRABOWSKA,
Wojciech WALOSZEK and Michał ZAWADZKI
Gdansk University of Technology, Department of Software Engineering,
ul. Gabriela Narutowicza 11/12, 80-952 Gdansk, Poland
e-mails: {kris, tegra, wowal, michawa}@eti.pg.gda.pl

Abstract. The paper presents research and development accomplishments achieved so far during work on a Knowledge Management Subsystem (KMS) for a e-health system called PIPS. The paper presents the Semantic Tools layer of KMS, and concentrates particularly on the PIPS Knowledge Inference Engine (KIE). The following issues are discussed: an analysis of freely available knowledge processors, conformance to existing Semantic Web standards, basics of an algorithm called Cartographer applied in KIE, results of application of KIE based on Cartographer in the PIPS Knowledge Base and perspectives for next stages of the project.

Introduction

PIPS (*Personalised Information Platform for Life and Health Services*) is an R&D integrated project carried out within the 6th Framework Programme of European Union, area *Information Society Technologies*, priority E-health. The project, whose duration spans from 2004 till 2007, is realized by a consortium consisting of 17 partners from Europe (from Poland: GUT and Atena Ltd.), Canada, and China. The main goal of the project is development of a distributed, Web-based platform that would support health care of EU citizens and would help them to keep a healthy life-style. The heart of PIPS are two activities related to development of the “intelligent” part of PIPS that consists of the Decision Support Subsystem (DSS) and the Knowledge Management Subsystem (KMS). Basics assumptions and background that are behind the both subsystem have been addressed in the previous paper [1]. To remind briefly: DSS consists of a set agents that are responsible for interaction with users (doctors, patient, or other citizens), and a database that contains, among others, information about patients (clinical records) in the form of so-called *virtual egos*. To fulfill the users’ demands or queries, or to react on some facts concerning patients reported by tele-medicine devices, some DSS agents (called *Knowledge Discovery Agents*, KDAs) communicate with the PIPS Knowledge Base (KB) managed by the Knowledge Management Subsystem (KMS). In its Knowledge Base (KB), KMS stores relevant knowledge as *ontologies* [2], general or especially suited for PIPS.

¹ Work partially sponsored by the 6. Framework Programme of EU, Contract No. IST-507019-PIPS

An essential part of KMS is the Knowledge Inference Engine (KIE) that is able to perform reasoning over the ontologies to infer new knowledge from the stored one. The knowledge is transferred from KIE to the KDAs according to a specific protocol and is further used by DSS to take appropriate decisions.

In this paper, we concentrate on the Knowledge Management Subsystem of PIPS, particularly on the Knowledge Inference Engine. The reason for that is that the authors of this paper were assigned a task of design and development of KMS that – according to the Description of Work for PIPS – could: (1) deliver trustworthy and dependable knowledge base covering broad range of knowledge on health and healthy lifestyle, (2) offer an efficient, scalable, flexible, distributed platform for managing knowledge base and ontologies, (3) be compliant with the Semantic Web emerging standards. In [1], we have formulated some problems to be solved in the PIPS KMS, like inferring new knowledge from Description Logics (DL) [3] ontologies or scalability of KMS with respect to the number of individuals stored in the KB. The rest of this paper describes how these problems have been solved in the first, prototype release of PIPS KMS. In Section 1 we present the architecture of Semantic Tools. In Section 2 Cartographer algorithm [4] for inferring over DL ontologies is briefly introduced. Section 3 concludes the paper with perspectives of new functionalities to be included into future releases of PIPS Knowledge Base.

1. Architecture of PIPS Knowledge Base

The PIPS Knowledge Base has been divided into two parts. *An ontological part* of the Knowledge Base holds data in a form allowing for inferences with respect to defined concepts. This part contains TBox and ABox data and is managed by Knowledge Inference Engine. *A data part* of the Knowledge Base (called DBox) holds numerical and string values of attributes of individuals and is managed by DBox Manager. From conceptual point of view, DBox is a part of ABox, but it has been separated from the rest of KB for efficiency reasons. Indeed, PIPS ontologies contain a lot of individuals with many numerical and string attributes that are not important for inference process but are important for KDAs as pure data. Storing them directly in ABox of the Knowledge Base would considerably increase the size of the database and affect performance. In the current version of the system, both parts of the Knowledge Base are populated exclusively from ontologies, but in future releases they will be also populated from external data sources via Syntactical Tools layer of KMS. Basic components of Semantic Tools for the PIPS Demonstrator are presented in Figure 1. A KDA passes its query to Query Manager. Query Manager does a preliminary analysis of the query and determines the class of the query. Four classes of queries have been identified: **TBox class queries** are related only to terminology (i.e. to TBox exclusively). These queries are directed to Knowledge Inference Engine. An example of such query is the subsumption problem, like: “Is *Pneumonia* an *IndustrialDis ease*” (*Pneumonia* and *IndustrialDisease* are concepts). **ABox class queries** correspond to queries about individuals with respect to terminology. These queries are also directed to Knowledge Inference Engine. An example of such a query is the retrieval problem, like: “Give all instances of *CertifiedDrug*” (*CertifiedDrug* is a concept). **DBox class queries** correspond to queries about individuals with respect to their attributes. An example of such a query is “Give *John Smith's age*” (*John Smith* is and individual and *age* is a numerical attribute). These queries are directed to DBox Manager. **ABox** –

DBox class queries correspond to queries about values of attributes of individuals with respect to terminology. An example of such query is “Give all instances of *Drug* that contain at most 2.5 grams of *paracetamol*” (*Drug* is a concept and *contains Paracetamol* is a numerical attribute). Execution of this kind of queries requires participation of both KIE and DBox Manager, because both ontological and data part of the Knowledge Base are involved. In each case, Query Planner prepares an execution plan for a query, which is then interpreted by Query Manager by issuing appropriate calls to KIE and DBox Manager. Returned results are merged by Query Integrator and passed to the calling KDA. The KAdmin and DBAdmin modules are responsible for creation and initialization of TBox, ABox and DBox databases to store an ontology, and are able to load an ontology into appropriate components of KB.

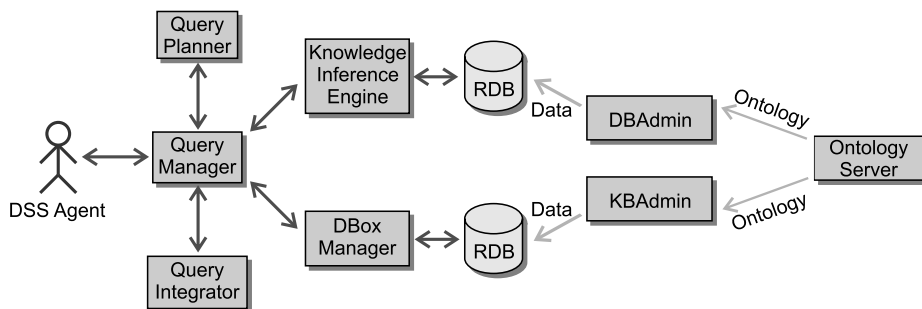


Figure 1. Components of PIPS Semantic Tools

2. The Algorithm

In this section we briefly introduce main ideas of Cartographer that constitutes an algorithmic basis for Knowledge Inference Engine in PIPS. Cartographer is an algorithm for processing DL ontologies and for reasoning over DL ontologies. Cartographer aims at storing in the knowledge base as many conclusions about concepts and individuals as possible. The conclusions can be quickly retrieved from the knowledge base in the process of query answering and remain valid due to the assumption that terminology (TBox) cannot be updated. By proper organization of the knowledge base the same conclusions can be applied to any number of individuals, facilitating efficient and scalable information retrieval and reducing the size of the knowledge base. The idea of Knowledge Cartography takes its name after a *map of concepts*. A map of concepts is basically a description of interrelationships between concepts in a terminology. The map is created in the course of knowledge base creation, i.e. during ontology loading. A map of concepts can be graphically represented in a form similar to a Venn diagram (see Figure 2).

Each new concept added to TBox divides the domain (denoted by T) into two disjoint regions: one that contains individuals that belong to the concept, and the other that contains individuals that do not belong to the concept (the rest of the world). In that way, after defining atomic concepts A , B , and C the domain has been divided into eight disjoint regions, numbered from 1 to 8 (see Figure 2a). Each region is assigned a corresponding unique position in a string of bits called *signature*. In that way, we can represent each concept by its signature with “1”-s at positions that correspond to regions

that the concept includes, and with “0”-s elsewhere. Signatures of complex concepts (like D and E in Figure 2a) are created by applying appropriate Boolean operators to signatures. Let us add two new axioms to our exemplary TBox (see Figure 2b). The first axiom states that A and B are disjoint, which eliminates regions 3 and 4 from the map (these regions become *unsatisfiable*, which means that they cannot have instances). The second axiom states that C is included by B , which additionally eliminates regions 5 and 8. Finally, we obtain four regions, renumbered as from 1 to 4 in Figure 2b. In the same way, each individual can be assigned a signature that defines its “region of confidence”, which actually represents what KB “knows” about the individual. The more “1”-s in an individual signature, the less KB “knows” about it. For instance, assume that we know that a new individual x is an instance of B . So, it is assigned the same signature as B : “0011”. At this point we do not know whether x is an instance of C . If – after some time – KB gains additional information that x is not an instance of C , its signature is changed to “0010” reflecting the fact that our knowledge about x has become more precise. Even this simple example illustrates remarkable potentials of Cartographer. Indeed, the Knowledge Cartography approach:

- allows for efficient handling of basic Description Logics inferences: subsumption, disjointness, equivalence, and satisfiability, as well as non-standard ones, like least common subsumer or most specific concept [3];
- allows for application of constructs postulated for expressive Description Logics that are useful for reasoning in real-life applications, like role chaining, role intersection, role complement, etc.;
- offers possibility to formulate complex assertions, e.g. $\neg C(x)$ or $A \sqcup B(x)$;
- enables to store inferred conclusions in a relational database, which is of crucial importance for scalability of KB;
- in a natural way supports three-valued logic (*true*, *false*, *don't know*) inherent to the *Open World Assumption* (OWA) in knowledge-based systems.

The last point needs some clarification. In databases, the *Closed World Assumption* (CWA) is followed. According to CWA, only those assertions (facts) are true (exist) that are stored in the database. Indeed, if we formulate any query on instances as an SQL statement, the result set will be either non-empty or empty. As a result the answer to any instance-related query will either be “true” or “false”. In contrast, according to OWA, if an assertion is not stored in the knowledge base, this assertion may be either true or false: the knowledge base simply “does not know” whether the assertion holds. Let us go back to the above example with instance x . After interpreting assertion $B(x)$, the KB still “does not know” whether x is an instance of C or not, so the answer to the query: “Is x an instance of C ?” should be “don't know”. Cartographer is able to respond in such a way because signatures of B and C partly overlap and x can lie in any region indicated by “1”-s in the signature of B . Full consequences of applying CWA instead of OWA may be fundamental for results of reasoning and detailed discussion is outside the scope of this paper. In the current implementation, Cartographer conforms to OWA only in some classes of queries.

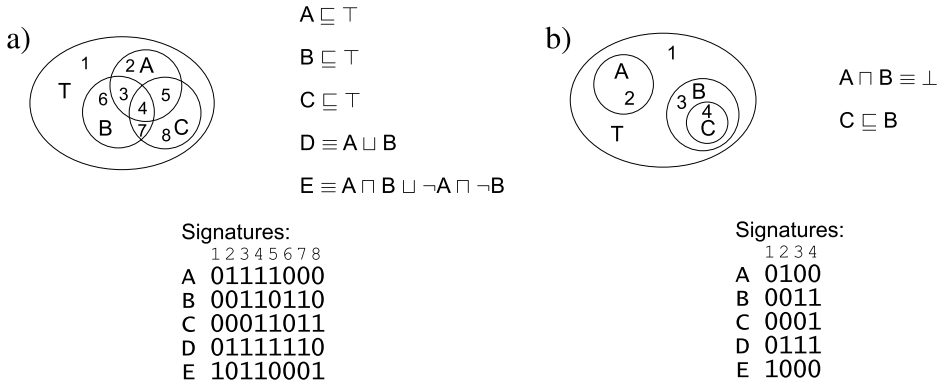


Figure 2. A map of 5 concepts (a), with two terminological axioms added (b)

The Cartographer-based KIE performance for ABox reasoning was compared with Jena 2 [5] and Racer [7] (FaCT [6] was not included in the tests because of its lack of support for ABox). Some results of the tests are presented in Table 1. The tests were performed on a PC with Celeron 2.4GHz and 256MB RAM. The back-end of KIE was a PostgreSQL v7 database. The main difference between analyzed reasoning algorithms is related to time of loading an ontology (TBox and ABox). The time of loading ontology is longer for Cartographer. In return a very short time of response is obtained. While Racer was unable to answer a query after 1000 individuals have been loaded to KB, our KIE could process the same query for 11000 individuals in 1.4 second. The results seem to be important for real-life applications, like PIPS, where changes in terminologies are infrequent and response time is much more critical for the overall performance of the system than ontology loading time.

Table 1. Results of efficiency experiments. Hyphens denote that the activity could not be completed within 2 hours.

Size of ABox	Loading time [s]			Query-processing time [s]		
	400	1000	3800	400	1000	3800
Jena	1	22	–	6	250	–
RACER	3	4	5	58	–	–
Cartographer	43	122	465	<1	<1	1

3. Summary

Capabilities of PIPS Knowledge Inference Engine and Knowledge Base have been verified and validated against first versions of PIPS ontologies concerning the problem of diabetes. They contain concepts of *Person*, *Food*, *Product*, *ClinicalRecord* and *Diabetes*, and their related subconcepts, axioms and assertions. They together form an integrated (via import mechanism) ontology of approx. 1000 concepts, 1000 assertions

and 4000 axioms. The ontologies have been used for the first release of PIPS system called Demonstrator. Experiments performed with Demonstrator showed that the Cartographer-based KIE fully meets DSS expectations. The PIPS ontology can be loaded within several minutes, which means that the forward-chaining of Cartographer is performed efficiently and the limitation that TBox cannot be modified on-line is not severe. Asks issued by KDAs can be processed in short time (fractions of a second to several seconds, depending on kind of query), and capability to respond to an ABox query does not depend on the size of ABox. These results encourage us to further enhance functionality of the engine. Directions of enhancement that are both interesting from research point of view and useful for PIPS system purposes include non-monotonic reasoning, multi-ontology queries [8] and trust issues [9]. Cartographer Approach to knowledge representation will be also further developed towards support of expressive DL constructs, particularly in querying, and towards uniformity in handling roles and concepts.

References

- [1] Goczyla, K., Grabowska, T., Waloszek, W., Zawadzki, M.: Issues related to Knowledge Management in e-health Systems, (in Polish). In: "Software Engineering – New Challenges", Eds. J. Gorski, A. Wardzinski, *WNT* 2004, Chap. XXVI, pp. 358-371.
- [2] Staab, S., Studer, R.: Handbook on Ontologies, *Springer-Verlag*, Berlin, 2004.
- [3] Baader, F.A., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, implementation, and applications, *Cambridge University Press*, 2003.
- [4] Goczyla, K., Grabowska, T., Waloszek, W., Zawadzki, M.: The Cartographer Algorithm for Processing and Querying Description Logics Ontologies. Lecture Notes in *Artificial Intelligence*, June 2005 (to appear).
- [5] A Semantic Web Framework for Java, <http://jena.sourceforge.net>
- [6] Horrocks, I.: FaCT Reference Manual v 1.6, August 1998, Included in FaCT archive from <http://www.cs.man.ac.uk/norrocks/FaCT>
- [7] Haarslev, V., Moller, R.: RACER User's Guide and Reference Manual, September 17, 2003, <http://www.cs.concordia.ca/haarslev/racer/racer-manual-1-7-7.pdf>
- [8] Goczyla, K., Grabowska, T.: Query reformulation in a distributed Description Logics knowledge base, (in Polish), National Conference on Databases Applications and Systems, 2005 (to appear).
- [9] Goczyla, K., Zawadzki, M.: Processing and inferring from knowledge of different trust levels, (in Polish), National Conference on Databases: Applications and Systems, 2005 (to appear).

Open Source – Ideology or Methodology?

Krzysztof DOROSZ and Sebastian GURGUL
AGH University of Science and Technology,
30-059 Krakow, Poland,
e-mails: {cypreess, gurgul}@student.uci.agh.edu.pl

Abstract. Come into being of complex FOSS projects, achieving success on the market is forcing to wonder on the method of their rising. For sure good software isn't able to arise without the usage of no production – methodology rules. It is appearing, that the present form of the Open Source movement has matured already enough, not to be call only the ideology, but to be called the balanced methodology of software development.

1. How were they Starting?

At American universities MIT, Cernegie Mellon, Berkeley and Stanford in sixtieth years of past age rose new hackers' culture, which began the movement of free software development. Within years this idea got stronger and she was evolving, gathering around oneself both masses of followers and opponents. Followers of FOSS software (Free/Open Source software) can see the chance in it for developing one's skills, producing of software adjusted to their needs, as well as yield possibility for the global exchange of production's experience. Enemies are accusing this approach of the business risk tied with openness of the source code and problems with deriving direct financial benefits from software as so.

As for now, two main and active trends of development of free software are FSF (Free Software Foundation¹) and OSI (Open Source Initiative²). Both approaches are coexisting in the one area of open software production. FSF assumes the freedom of using, acquainting oneself with the architecture, modification and redistribution of the software with source code of it. However, OSI is emphasizing practical benefits derived from opening the source code, though isn't taking moral aspects of such a approach up intensively. For many years both philosophies are existing side by side and they are efficiently promoting proclaimed ideas in spite of some ideological differences.

2. Whom are they with?

Open Source is the methodology oriented to people. Group work is nothing new or revolutionary, what a way of controlling of these groups and specific communication's methods, which are very characteristic for Open Source projects. Participants in

¹ <http://www.gnu.org>

² <http://www.opensource.org>

undertakings bound with production of open software are descended from various groups: these are volunteers developing software only for pleasure, companies taking up sale of systems based on the Unix (RedHat, Novell), companies, for whom OSS applications are main products (AB, Trolltech), companies in which the precise binding between sale of OSS software and sale of equipment is existing (IBM, HP), governments of states and towns (Germany), scientists and the academic community.

Contribution of each groups to specific OSS products is diversified and depends on specified group involvement and its possibilities. This contribution sometimes means money, sometimes concrete software work, but sometimes means merely role of the project's commentator. Three groups are participating directly in the process of software production: programmers, testers and users.

It is possible in developers' group to notice some distinctive features. They are usually intelligent and bright people. Their effort put on to development of FOSS projects is tied with noticeable lack of concrete functionality in existing software.

Testers are fulfilling in the manufacturing process of FOSS extremely significant role. The FOSS approach to the problem of the improvement programming in reliability can be briefly characterized with words:

"As more people test the software in real-world environments, more discrepancies between the abstract model and the world are found."

Frequent editions of applications are the direct effect from such kind of approach, of whom the consequence is quick delivering of new or adjusted functionality to the user.

The average user of FOSS software is able to find and to identify software errors thanks to the fact, that programmers are posing users' big part in every popular project (prone to express one's opinions and being helpful during errors finding). It is huge advantage of FOSS projects over competitive solutions. Being intelligent and knowing "software problems" user is the valuable purchase for every project.

Open Source programmers don't have to search and to employ testers for projects. It is coming true better considerably giving software in the end user's hand and making the beta-tester from him. It is possible thanks to the fact, that average level of knowledge of users utilizing Open Source software is much higher than of average users of commercial programs.

3. How is it Happening?

"We reject kings, presidents and voting. We believe in rough consensus and running code." (Dave Clark)

Lack of the central authority inside the project is the distinctive feature of the organizational structure of Open Source projects, which would have strength of persuasion being enough to force anybody to do anything. The organizational structure of Open Source projects is strongly decentralized. Participants are independent and to a large degree capable to work at these aspects, which are interesting them. However, it doesn't mean any chaos.

Every project must have place for the leader, who is co-ordinating and integrating the whole process of software development. What is extremely important is the fact, that his authority always bases on confidence got thanks to his competences. It is

possible with this approach to emphasize four distinctive features: decentralised cooperation, the lead based on confidence, motivations and asynchronous communication.

Decentralized Cooperation

Open Source projects are often perceived as anarchical. This opinion is rising probably of opinions, which they are claiming that programmers are avoiding and they don't like the documentation and bureaucratic rules. This is obvious simplification. In real conditions, despite the fact that FOSS programmers have big independence, their actions are undergoing continuous control of the persons responsible for whole of the project.

"It is a chaos with some external constraints put on it... it allows a chaos, but at the same time it has certain built-in things that just make it very stable." (Linus Torvalds)

As the project is gathering bigger participants' group round itself, this control is becoming more and more necessary. The group would not be capable to collaborate at big tasks without the determined lead. People engaged in FOSS projects are working on these bits of the code which are interesting them really. Despite this fact, the problem of integration of the own code always appears with parts created by other people. This task can become more difficult, if the more bigger developers' group is participating in it. It is the result, that usually has to a person exist who will undertake the role of the code integrator and will be responsible for taking the decision which and when it is supposed to be found in the main distribution up. The person who will be in condition to undertake this challenge is becoming the informal leader of the project (are existing FOSS projects without the determined leader and their development is based on supervision of the committee consisting from leaders' group).

Organizational structure of the group in the FOSS project (Figure 1) assumes 4-levels hierarchy of functions fulfilled in the project. Person (or persons' group) responsible for the project is strictly collaborating with the group (usually, few) of persons knowing the architecture of created software perfectly. These persons are responsible for co-ordinating work in the distributed structure and for integration of the code created by "rank-and-file" programmers. Successive versions of software are used and tested by users' community and testers of the product.

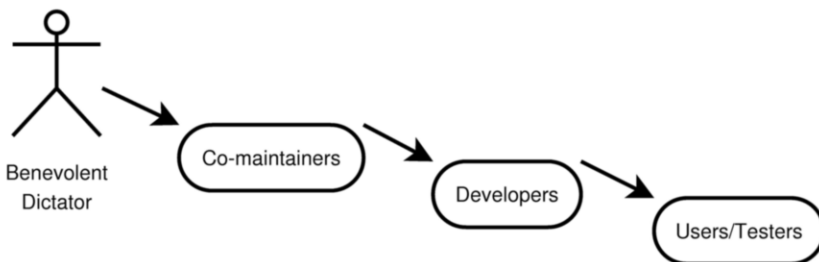


Figure 1. Organizational structure of the group

Lead Based on Confidence

The leader of the project should have the possibility (peaceful) of influencing on project's participants. The lead in Open Source projects is based on the reputation and trust in the concrete person, what's usually depends on competence and charisma of

this individual. The founder of the project usually becomes his initial leader. This situation is subject to change, if the project would gather the large enough community round itself. The leader's change for the person with better competences is the natural phenomenon then (in accordance to the principle of the natural selection).

Internal Motivation

The motivation is the main factor having influence on people's efficiency, what is giving out in the form of the quality of the product directly. In case of Open Source programmers the need is the main supporting factor, having influence on useful software creating.

It is also the important supporting factor, when important work is being executed. Open Source projects are attracting many programmers this way, because they are always the challenge merged with producing something unconventional.

Asynchronous Communication

Communication is very characteristic in Open Source projects, because of the geographical distraction of participants in the project. So almost all projects are using mailing lists as the main method of communication. It has peculiar results – it is making possible to acquaint himself with opinions and the whole community's conclusions and is creating unusual kind of documentation of the project. Direct meetings of participants in the project very often don't happen at all.

Communication between users and programmers is going through the company's department of marketing in the majority of commercial projects. Open Source projects are in this respect entirely different. Users often have quite immense knowledge in these projects, so their communication with programmers is most beneficial and is bringing advantages for both of sides. Such a solution is a motivation for the whole community of the project and is contributing to efficient project's development – what's important, in direction anticipated by the user.

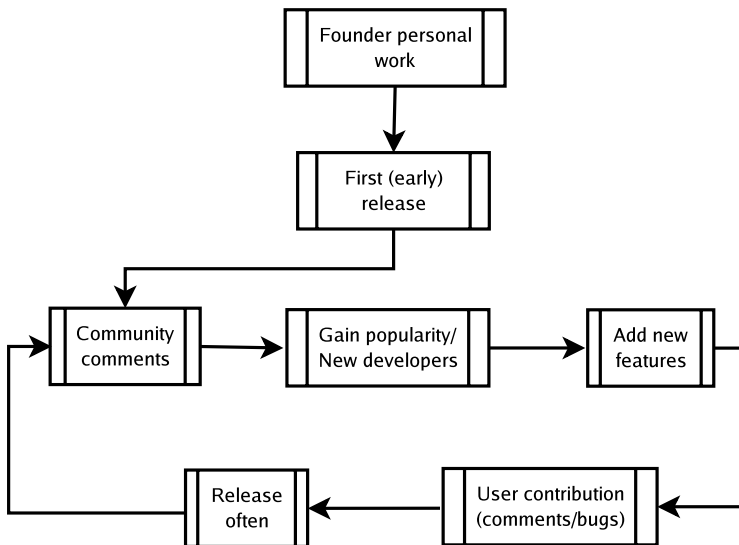


Figure 2. Software processes

Life cycle of the FOSS project (Figure 2) comprises whole of the process of producing and maintaining the program. After the first, early edition the product is entering to the stage of cyclic development. This stage assumes continuous cooperation with users in order defining functional requirements of the project, widening the community participating in the project and putting changes proposed by users as soon as possible into practice and delivering the next versions of product, which would not necessarily be entirely stable, to community. Above described life cycle of the project is defining the process of project rising on the programming level.

Other, possible to define approach, is 6-stage's model, taking into consideration market position of the project and project's existence with consideration of other projects (also commercial projects) presented in the Figure 3.

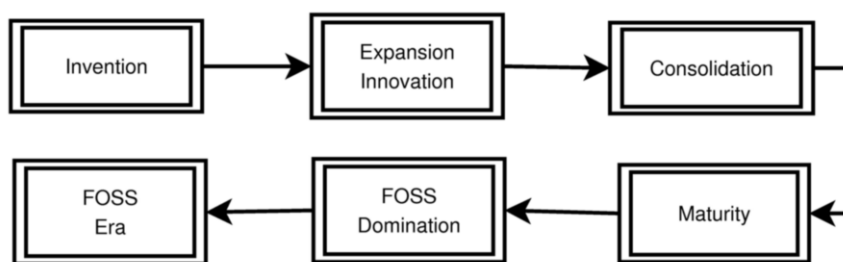


Figure 3. Life cycle of the project

Invention

Every project comes into being as a result of the concrete need for the human of creating more effective solutions and the automation of daily activities.

Expansion and Innovation

The investor finances initial stages of the project development, is existing in commercial projects. Lack of financial support is the reason for the fall many of FOSS projects, in most cases at this stage of development. Currently, many big companies can perceive chance in software creating using this method and they are sponsoring and co-creating FOSS projects.

Consolidation

The FOSS methodology is trying to exclude rivalry between similar projects. In the Consolidation phase, evidently better projects are assimilating other similar solutions. It makes possible exploitation / taking over of the individual projects communities and theirs best achievements in this discipline. As a result of it, the one collective project is rising, concentrating all best features in oneself and having a chance for entering into the struggle with commercial solutions.

Maturity

The project is mature so much in this phase yet, that changes are being relatively rarely introduced into functionality. Software is operating stably already, the ratio of the critical errors quantity is relatively low. At the same time, the project is determining

standards in the given discipline, and since the level of the project advancement is closest to these standards, creating the new product with similar functionality is extremely difficult.

FOSS Domination

This phase binds with the fast increasing number of the application users. Reasons are simple, at this stage it is already stable, reliable, safe product which is delivering functionalities anticipated by users and additionally free from charges and well documented.

FOSS era

Final success of the project means full market domination, which is bound with pushing all other solutions (also commercial) aside to market niches or to their total elimination.

4. What are they Reaching?

Since many years FOSS projects are proving their usefulness. Linux, Apache, Bind, OpenSSH these are products which enabled the global network development and for today they are base for this network. Theirs quality and benefits derived from the approach based on the open source code are persuading large software houses today to invest in existing FOSS projects development (e.g. RedHat). What's more, the same manufacturers more often taking a decision for opening the code of theirs products (e.g. Solaris, Eclipse) and their more faraway development in the spirit of the Open Source idea. It is proving, contrary to generally accepted views, that FOSS is also creating possibilities of making money on created software. It is FOSS programmers' huge success, that deriving pecuniary advantages in the modern FOSS model is taking place not by making secret of the source code of applications locking future development, but through assuring technical support and professional training for users (mainly for business users).

References

- [1] Mockus, R.T., Fielding, J., Herbsle, A Case Study of open source Software Development: The Apache Server (ICSE 2000)
- [2] D.M. Nichols, B. Michael, The Usability of Open Source Software, (First Monday 2002)
- [3] K. Wong, P. Sayo, Free/Open Source Software – A general introduction, (UNDP-APDIP 2004)
- [4] F. Lehmann, FOSS developers as a social formation, (First Monday 2004)
- [5] E.S. Raymond, The Cathedral and the Bazaar, (Thyrus Enterprises 2000)
- [6] A.J. Craig, The Care and Feeding of FOSS (2004)
- [7] J. Lerner, J. Tirole, The Simple Economics of Open Source (2000)

This page intentionally left blank

Author Index

Adamus, R.	245	Lyskawa, H.	221
Begier, B.	15	Madeyski, L.	113
Białas, A.	99	Małek, J.	406
Bluemke, I.	160	Markowski, P.	39
Bobkowska, A.	75	Michlmayr, M.	3
Brochocki, M.	221	Nalepa, G.J.	294, 330
Bukowy, M.	27	Nawrocki, J.	319, 400
Chrząszcz, J.	412	Nikolov, D.	149
Czyrnek, D.	209	Nowak, T.	233
Czyz, M.	135	Nunn, D.	27
Demuth, A.	400	Olek, Ł.	319
Derezińska, A.	160	Pietrzak, B.	353
Dorosz, K.	424	Podyma, M.	305
Dorsz, M.	400	Reszke, K.	75
Finch, S.	27	Rogus, G.	365
Fitrzyk, R.	406	Sacha, K.	381
Funika, W.	184	Samolej, S.	194
Gąsienica-Samek, A.	412	Schubert, A.	412
Głowacki, E.	245	Serafiński, T.	245
Głowacki, M.	233	Skitał, Ł.	149
Goczyła, K.	271, 418	Skrzyński, P.	87
Godowski, M.	209	Słota, R.	149
Gorawski, M.	49	Śmiałek, M.	341
Graboń, M.	406	Stachowicz, T.	412
Grabowska, T.	271, 418	Subieta, K.	245
Gurgul, S.	424	Surkont, M.	406
Hnatkowska, B.	63, 124	Szejko, S.	221
Huzar, A.	406	Szmuc, T.	v, 194, 365
Huzar, Z.	63	Szpyrka, M.	294, 394
Jadach, M.	124	Szymański, K.	259
Janik, A.	184	Trawiński, B.	305
Jarzab, M.	172	Trocki, K.	135
Kaliś, A.	406	Turek, M.	87
Kardas, A.	341	Tuzinkiewicz, L.	63
Kitowski, J.	149	Valenta, M.A.	283
Klimek, R.	87	Waloszek, W.	271, 418
Kmiecik, T.	283	Walter, B.	353
Kosinski, J.	172	Wierzchon, L.	389
Koźlak, J.	259	Wilder, L.	27
Kozłowski, W.E.	221	Woroszczuk, P.	406
Król, D.	305	Zawadzki, M.	271, 418
Ligeża, A.	330	Zieliński, K.	v, 172
Łuszpaj, A.	259	Zygmunt, A.	259, 283

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank